Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ADSL	Asymmetric Digital Subscriber Line
ALSA	Advanced Linux Sound Architecture
API	Application Programming Interface
CA	Conditional Access
CPU	Central Processing Unit
CTI	Computer Telephony Integration
DNG	Delivery Network Gateway
DSL	Digital Subscriber Line
DSLAM	Digital Subscriber Line Access Multiplexer
DTS	Digital Theater Systems
DVB	Digital Video Broadcasting
DVI	Digital Visual Interface
DVR	Digital Video Recorder
EDV	Elektronische Datenverarbeitung
EPG	Electronic Program Guide
eSATA	External Serial Advanced Technology Attachment
ETSI	European Telecommunications Standards Institute
GPU	Graphics Processing Unit
HDMI	High Definition Multimedia Interface
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IR	Infrarot

ISO	International Organization for Standardization
ITU	International Telecommunication Union
MAC	Media Access Control
MFC	Microsoft Foundation Class
miniATX	miniAdvanced Technology Extended
MPEG	Moving Picture Experts Group
OSI	Open Systems Interconnection Reference Model
OVSt	Ortsvermittlungsstelle
PDA	Personal Digital Assistant
PPP	Point-to-Point Protocol
PPPoE	Point-to-Point Protocol over Ethernet
PVR	Personal Video Recorder
SCART	Syndicat des Constructeurs d'Appareils Radiorécepteurs et Téléviseurs
SDTV	Standard-definition television
UML	Unified Modeling Language
URL	Uniform Resource Locator
USB	Universal Serial Bus
usw.	und so weiter
VDSL	Very High Speed Digital Subscriber Line
VLAN	Virtual Local Area Network
WAN	Wide Area Network
WLAN	Wireless Local Area Network
WWW	World Wide Web
XML	Extensible Markup Language
z.B.	zum Beispiel

1 Einleitung

Im einleitenden Kapitel der Diplomarbeit werden die Motivation und die Zielsetzung besprochen. Außerdem erfolgt ein kurzer Überblick über die Kapitel der Arbeit.

1.1 Motivation

Schon längst ist das Massenmedium Fernsehen aus den weltweiten Haushalten nicht mehr wegzudenken. Immer breitere Programmangebote werden vom Anbieter für den Konsumenten bereitgestellt. Ermöglicht werden die Fortschritte durch die zunehmende Digitalisierung des Fernsehnetzes, wodurch moderne Kompressions- und Übertragungsverfahren eingesetzt werden können.

Doch nicht nur das Medium Fernsehen erlebte in den letzten Jahren eine rasante Entwicklung. Das Internet mit seinen nahezu unbegrenzten Reichtum an Informationen und Wissen durchlief seit seiner Geburt eine regelrechte Wachstumsexplosion. Durch den Einzug des Internet's in elektronische Geräte, die im Alltag vieler Bürger nicht mehr wegzudenken sind, nimmt das Internet einen immer höheren Stellenwert ein.

Diese Gründe brachten Telekommunikationsunternehmen aber auch Privatentwickler auf die Idee, Vorteile beider Informationsquellen zu vereinen und als Gesamtpaket dem Konsumenten anzubieten – IPTV und Internet-TV. Gerade im nicht-kommerziellen Bereich stehen dem Anwender durch quell-offene Softwarelösungen in Form von Media Center nahezu unbegrenzte Möglichkeiten, das System an individuelle Bedürfnisse anzupassen und es so zu einer Multimediaplattform mit interaktiven Elementen zu verwandeln. So lassen sich über Programmschnittstellen der quell-offenen Softwarelösungen eigene Funktionalitäten implementieren und die Oberfläche der Anwendungen selbst gestalten.

1.2 Zielsetzung

Die vorliegende Arbeit befasst sich im weitläufigeren Rahmen der Aufgabenstellung mit der einleitenden Beschreibung von IPTV und Internet-TV. Es werden dabei unter anderem technische Grundlagen im Bereich IPTV, speziell aus dem ETSI-Standard DVB-IPI, sowohl für Internet-TV und der dahinter liegenden Technologie Internet-Streaming vorgestellt und typische Dienste erläutert. Des Weiteren werden aktuelle technische Ausstattungen von Endgeräten im kommerziellen IPTV- und Internet-TV-Bereich sowie Ausstattungen von nicht-kommerziellen Softwareanwendungen vorgestellt.

Nach dem kurzen Überblick über IPTV und Internet-TV wird sich der Aufgabenstellung weiter angenähert. Dabei sollen Grundlagen, Aufbaukonzepte und Anforderungen an

Home Theatre Personal Computer (HTPC) vermittelt werden. Im Rahmen dieser Annäherung soll die Vorstellung des Xbox Media Center's erfolgen.

Für das Hauptziel wird die Integration eines CTI-Telefonclients in das bestehende und quell-offene Xbox Media Center, im speziellen eine Anrufsignalisierung mit Anrufsteuerung, ein Telefonbuch und ein Gesprächszeitzähler beschrieben.

1.3 Kapitelübersicht

Die Diplomarbeit gliedert sich in sechs Kapitel.

Kapitel Eins stellt die Motivation und das Ziel der Arbeit dar.

Im zweiten Kapitel werden wichtige Definitionen geklärt und technische Grundlagen zu IPTV und Internet-TV geschaffen.

Das Kapitel drei befasst sich mit der Präzisierung der Aufgabenstellung und der damit verbundenen Vorstellung des Xbox Media Center's.

Um die Vorstellung des Systemkonzeptes befasst sich das vierte Kapitel. Anwendungsfälle und Anforderungen an das System werden definiert und in Abhängigkeit davon die benötigten Hardware- und Softwarekomponenten vorgestellt. Grundlagen zu benötigten Technologien werden ebenfalls erläutert.

Das fünfte Kapitel befasst sich mit der Implementierung. Hier werden umfangreiche Erläuterungen zu Installationsschritten, Konfigurationseinstellungen und Quelltextabschnitten gegeben.

Eine Bewertung der Ergebnisse wird im sechsten Kapitel vorgestellt. Hierbei wird die Umsetzung der CTI-Funktionalität im Xbox Media Center bewertet und kritisch hinterfragt.

Am Ende schließt sich das siebte Kapitel mit den Ausblick auf kommende Versionen des Xbox Media Center's und eventuellen Verbesserungen an.

2 Grundlagen zu IPTV und Internet-TV

In diesem Kapitel soll die Grundlage für weiterführende Überlegungen und Implementierungen geschaffen werden. So werden wichtige Begriffe im Bereich IPTV und Internet-TV definiert und technische Zusammenhänge geklärt. Häufige Anwendungsszenarien und Architekturmodelle werden außerdem beschrieben und vorgestellt.

2.1 Begriffsdefinition und wesentliche Unterschiede

Internet Protocol Television, kurz IPTV, definiert Multimediadienste, die über Internet Protocol-basierte Netze (IP-Netze) übertragen werden. Dabei können sowohl Audio, Bilder, Fernsehen und Video als Dienste bereitgestellt werden. Diese Begriffsdefinition stammt von der ITU. Der Begriff IPTV wird allerdings nicht einheitlich verwendet. So definiert der deutsche IPTV-Verband in seiner Satzung IPTV als die Übertragung von Bewegtbildern mit Hilfe des IP-Protokoll's unter Verwendung beliebiger Endgeräte und in aller Formen IP-fähiger Netze. Unter Formen wird hier ein geschlossenes Datennetz oder das öffentliche Internet verstanden. Wobei man bei Letzteren von Internet-TV oder Web-TV spricht. [1]

Ein wesentliches Merkmal von IPTV ist die geleitete Ende-zu-Ende-Verbindung mit dem IPTV-Anbieter. Dieser stellt dem Verbraucher ausgewählte Inhalte zur Verfügung. Der IPTV-Anbieter gewährt in geschlossenen Netzen eine Mindestbandbreite für eine sichere und zuverlässige Dienstfunktionalität. Hingegen wird bei der Zustellung von Videoinhalten aus dem WWW über das öffentliche Internet von keinen Anbieter eine Übertragungsqualität gewährleistet. [2]

Auch gibt es im Bereich der Endgeräte grundsätzliche Unterschiede zwischen IPTV und Internet-TV. Während Internet-TV über jedes Endgerät in Verbindung mit einen Internetanschluss und einer Audio-Video-Ausgabeeinheit empfangbar ist, bedarf es bei IPTV neben dem Internetanschluss eines vom Anbieter freigegebenen Endgeräts, einer Set-Top-Box. Meist sind die Programmangebote der Anbieter aus urheberrechtlichen Gründen auf eine bestimmte Art der Ausgabe beschränkt und so auf eine bestimmte Set-Top-Box zugeschnitten. [3, 4]

1 vgl. http://de.wikipedia.org/wiki/IPTV, Stand: 04.10.2010

2 vgl. http://de.wikipedia.org/wiki/Internetfernsehen, Stand: 04.10.201

3 vgl. http://de.wikipedia.org/wiki/Internetfernsehen, Stand: 04.10.201

4 vgl. http://de.wikipedia.org/wiki/Internet_Protocol_Television, Stand: 04.10.2010

2.2 Angebotene Dienste im Bereich IPTV und Internet-TV

Hinter IPTV verbirgt sich im klassischen Sinne eine kontinuierliche Ausstrahlung von Programminhalten. Interessante Zusatzoptionen und eine Vielzahl an Funktionen machen IPTV zu einer ernstzunehmenden Alternative gegenüber herkömmlichen unidirektionalen Empfangstechniken. Grundsätzlich werden beim typischen IPTV-Angebot die Dienste zwischen Live Media Broadcast (LMB) und On-Demand unterschieden. Während bei LMB oder auch Live-TV eine kontinuierliche Ausstrahlung von Inhalten erfolgt, entscheidet der Kunde bei On-Demand, wann er die Inhalte konsumieren will. Vergleichbar ist der Dienst mit einer Videothek. Nur muss der Kunde für die Ausleihe nicht mehr das Haus verlassen. In diesem Fall spricht man von einer Online-Videothek, also Video-On-Demand (VoD). Solche interaktiven Dienste bieten dem Zuschauer eine aktive Teilnahme am Programmgeschehen an. Zusatzoptionen wie Time-Shifting (zeitversetztes Fernsehen), EPG mit Aufnahmefunktion, Teletext oder Untertitel vervollständigen das Angebot. Meist werden dem Anwender von Anbietern, wie der Deutschen Telekom oder Alice, LMB-Dienste und VoD-Dienste als bezahlungspflichtiges Paket unter Einhaltung von Qualitätskriterien bei der Datenübertragung als IPTV-Paket verkauft.

Weitere Film- und Serienangebote sind bei Anbietern in TV-Archiven, den sogenannten Online-Mediatheken, zu finden. Kürzlich ausgestrahlte Filme, Nachrichten, Reportagen oder Serien können hier kostenfrei abgerufen werden. Online-Mediatheken erfahren im Bereich Internet-TV einer immer größeren Bedeutung. Während solche Mediatheken oft kostenfrei dem Anwender zur Verfügung stehen, bieten VoD-Portale außerhalb eines IPTV-Pakets hingegen Inhalte gegen Bezahlung an. Da hier der qualitätsgerechte Transport zum Anwender nicht durch das Portal gewährleistet werden kann, spricht man auch hier von Internet-TV.

2.3 Technische Grundlagen von IPTV

Wie schon im einleitenden Kapitel erwähnt, basiert die Übertragung der Signale für den Konsum von Fernsehen und anderen Diensten auf dem IP-Protokoll (IP). Im einfachsten Sinne versteht man darunter, dass die Videobilder in ein aktuell gängiges Format kodiert und komprimiert werden. Für den anschließenden Transport durch das Internet zum Empfänger wird das komprimierte Datenpaket mit einen IP-Rahmen versehen und so das Routen durch das Netz ermöglicht.

Speziell im kommerziellen Bereich ist eine einheitliche Standardisierung für den Anwender von besonderer Bedeutung. Wie in anderen informationstechnischen Bereichen wird auch hier an einen einheitlichen Übertragungsstandard entwickelt. Der von der ETSI entwickelte DVB-IPI-Standard vereint die aus der Satelliten-, Kabel- und Terrestrischen Übertragungstechnik eingeführten Komponenten aus den DVB-Systemen mit Internet-spezifischen Anforderungen. Ein Übertragungsstandard bringt den Vorteil beim Verbraucher mit sich, bei einen möglichen Wechsel des Anbieter's ein- und dieselbe Set-Top Box als Empfangsgerät verwenden zu können, sofern die Box diese Technik

unterstützt. Auch kann durch technisch bedingte Veränderungen die Software ohne Probleme durch Softwareupdates angepasst werden. Anbieter stellen diese Updates dem Verbraucher zur Verfügung. Öffentlich rechtliche Sender wie die ARD oder ZDF setzen bei der Ausstrahlung ihres unverschlüsselten Fernsehprogramms im VDSL-Netz der Deutschen Telekom (Telekom Entertain) und von Arcor auf diesen Standard [5].

2.3.1 Standardisierung

Voraussetzung einer lokalen, regionalen, nationalen oder internationalen Übertragung ist eine Standardisierung. Speziell in der Kommunikationstechnik werden hauptsächlich Dienste, Nutzer-Netz-Schnittstellen, Netz-Netz-Schnittstellen und Netzleistungsmerkmale standardisiert. Allerdings definieren Standards selten die optimale technische Lösung. Vielmehr sind Standards als Empfehlung anzusehen und dienen als Kompromiss zwischen Interessengruppen. [6]

Die ITU und deren Organe koordinieren und fördern die Entwicklungen der weltweiten IPTV-Standardisierungen unter Berücksichtigung der an den Standards arbeitenden Organisationen. [7]

Als möglicher und bevorzugter Übertragungsstandard für IPTV kommt derzeit DVB zum Einsatz. Der für IPTV herausgegebene und herstellerunabhängige ETSI-Standard DVB-Internet Protocol Interface, kurz DVB-IPI, spezifiziert Verfahren für den Transport von DVB-Diensten über IP-basierte Netze. Im Standard werden kritische Elemente des IPTV-System's betrachtet und Lösungen zur Standardisierung für Anwendungsfälle, Funktionen und Schnittstellen für Interoperabilität zwischen Geräteherstellern, Netzbetreiber und IPTV-Inhaltsanbieter vorgestellt. [8]

Die IETF definiert in ihren Internet-Drafts unter anderem die Implementierung eines Fehlerkorrekturverfahrens im DVB-IPI Standard oder die Medienzugriffskontrolle von IPTV-Diensten. [9]

---

5 vgl. http://de.wikipedia.org/wiki/DVB-IPI, Stand: 05.10.2010

6 https://www.telecom.hs-mittweida.de/index.php?eID=tx_nawsecuredl&u=0&file=fileadmin/verzeichnisfreigaben/telecom/winkler/vorlesungen/standardisierung.pdf&t=1296814808&hash=eda71254b4cde2c8124c5477e3e28437, Stand 03.02.2011

7 vgl. http://www.itu.int/en/ITU-T/gsi/iptv/Pages/default.aspx, Stand: 05.10.2010

8 vgl. http://www.etsi.org/WebSite/homepage.aspx, Stand: 05.10.2010

9 vgl. http://www.ietf.org/, Stand: 05.10.2010

2.3.2 IPTV-Architektur nach ITU-T H.720 [10]

Aufbauend auf den folgenden Überlegungen soll die IPTV-Architektur von der ITU als Grundlage dienen.

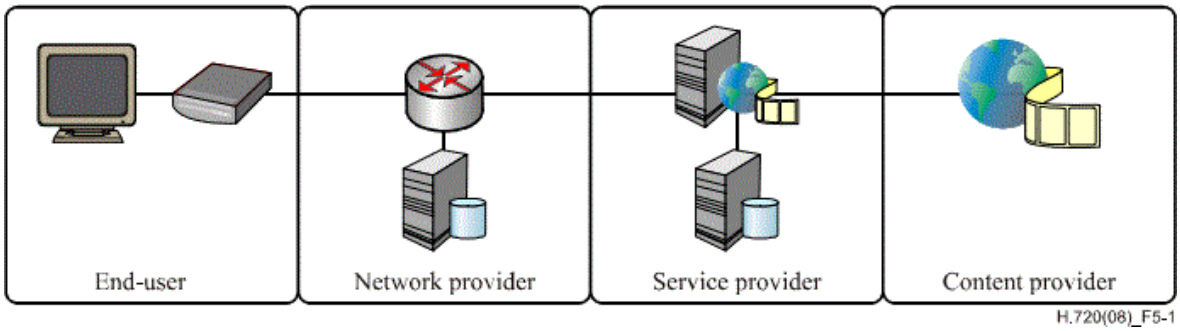

Abbildung 2-1: IPTV-Architektur nach T-REC-H.720

Abbildung 2-1 zeigt die für die Beschaffung eines Dienstes an den Konsumenten involvierten Systeme. Ein System erfasst hierbei alle Komponenten, die an der Beschaffung des Dienstes beteiligt sind. Insgesamt sind für die Zustellung vier Systeme erforderlich.

Der Content Provider (CP) besitzt oder ist lizenziert dazu, Inhalte beziehungsweise Inhaltsanlagen zu verkaufen. Daraufhin ist der CP die Hauptquelle des Endnutzer's.

Für die Verpackung von Inhalten in einen vom Endkunden beauftragten Dienst ist der Service Provider (SP) zuständig. Verschiedene Arten von SP können für DVB über IP relevant sein – Internet Service Provider (ISP) und Content Service Provider (CSP). Im Zusammenhang mit DVB-IPI-Diensten erwirbt oder lizenziert sich der CSP Inhalte vom CP und verpackt diesen in einen Dienst. Der ISP bietet die für den Dienst erforderlichen technischen Leistungen an.

Um die zeitnahe Zustellung kümmert sich der Network Provider (NP) . Das dahinterstehende System kombiniert Zugriffsnetzwerke, Kern- oder Backbone-Netzwerke unter Verwendung einer Vielzahl an Netzwerktechnologien.

Das Nutzerendgerät und Endsystem ist ein Teil der IPTV-Architektur und wird im End-User (Endnutzer)-Bereich eingeordnet. Ein Endsystem ist eine einzelne oder eine Menge an Verbrauchervorrichtungen. Es umfasst alle Punkte, ausgehend vom DNG bis hin zum Display.

2.3.3 Einblick in den DVB-IPI-Standard

Mit dem Standard werden Spezifikationen für die Zustellung von DVB-Diensten über IP-basierte Netze definiert. Dabei wird das komplexe System aus verschiedenen

10 vgl. ITU-T H.720 „Overview of IPTV terminal devices and end systems“

Blickwinkeln betrachtet. Im speziellen werden die an einer Dienstzustellung beteiligten Netzwerkschichten im Schichtenmodell dargestellt und die Schnittstellen im Heimreferenzmodell spezifiziert. Ziel des Standard's ist es, die einzelnen Systeme zu einen funktionierenden Gesamtsystem unter Berücksichtigung diverser Qualitätsanforderungen zu vereinen. Qualitätsanforderungen werden unter dem Begriff QoS [siehe Glossar] zusammengefasst.

2.3.3.1 DVB

Dem europäischen DVB-Projekt haben sich bis heute über 270 Mitgliedsfirmen aus Programmanbietern, Geräteherstellern, Netzbetreibern und Behörden, mit dem Ziel der Entwicklung eines europäischen Standards für Digitalrundfunk, Digital-TV, multimediale Dienste und interaktive Verteildienste, angeschlossen. Normierungsorganisationen wie die ETSI und CENELEC sind durch Kooperationsverträge für die entstehenden technischen Spezifikationen zuständig. [11]

Aus technischer Sichtweise werden bei DVB Verfahren zur Übertragung von digitalen Inhalten über verschiedene Übertragungswege durch digitale Technik bezeichnet. Inhalte können demzufolge über bekannte Übertragungswege wie Breitbandkabelnetze (DVB-C), Satelliten (DVB-S/S2), terrestrisch (DVB-T) oder über paketvermittelte Netze, man spricht von IP-Netzen, (DVB-IPI) verbreitet werden. [12]

Jeder dieser Übertragungswege stellt an das Signal unterschiedliche Anforderungen für eine qualitativ hochwertige digitale Übertragung. Da die Übertragungswege durch technisch bedingte Grenzen in Bandbreite und Datenrate unterschiedlich gekennzeichnet sind, mussten Übertragungsverfahren an den bestimmten Übertragungskanal angepasst und standardisiert werden.

2.3.3.2 Das Schichtenmodell [13]

Aus der Sichtweise eines Schichtenmodells kann die Zustellung eines Dienstes über das IP-Netzwerk beschrieben werden.

Das Modell in Abbildung 2-2 zeigt die relevanten Schichten für Dienste über das IP-Netzwerk und den dazugehörigen Peer-to-Peer Informationsfluss zwischen den verschiedenen Schichten. Zwischen den Schichten stellen verbindungsorientierte und verbindungslose Kommunikationsprozesse die Verbindung zwischen zwei Punkten (Data Link) oder über Internetnetzwerke (Network) her. Bei einem Peer-to-Peer-Netz sind alle

---

11 vgl. http://de.wikipedia.org/wiki/Digital_Video_Broadcasting, Stand: 06.10.2010

12 vgl. http://www.itwissen.info/definition/lexikon/digital-video-broadcasting-DVB.html, Stand: 06.10.2010

13 vgl. ETSI TR 102 033 (V1.1.1) „Digital Video Broadcasting (DVB); Architectural framework for the delivery of DVB-services over IP-based networks“

Komponenten in einem Netzwerk gleichberechtigt und können sowohl Dienste in Anspruch nehmen als auch Dienste bereitstellen.

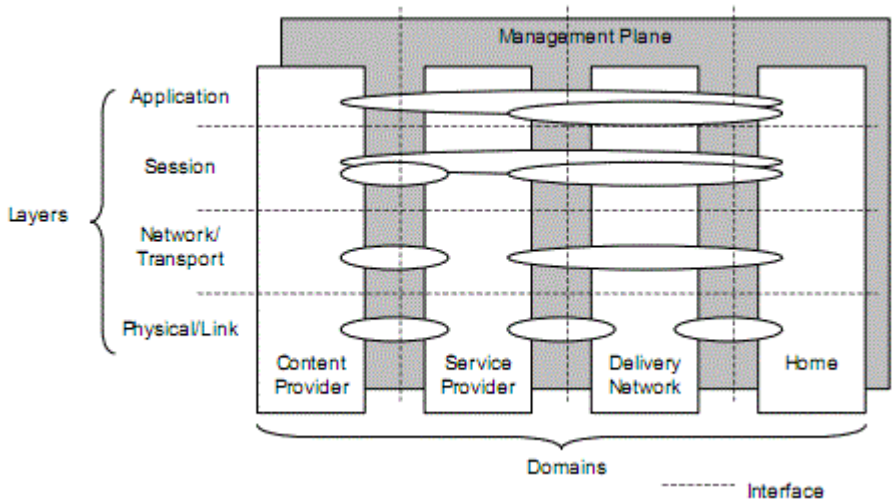

Abbildung 2-2: Einordnung der Dienstzustellung im Schichtenmodell

Jede Kommunikationsprotokollschicht präsentiert nur ein Service Interface (Dienstschnittstelle) mit versteckten Implementierungsdetails zur darüber liegenden Schicht. Dies bringt den Vorteil von reduzierten Kosten und geringerer Komplexität. Die Management Plane (Managementebene) ist für die Verwaltung und Kontrolle zuständig.

2.3.3.2.1 Physical (Bitübertragung) und Data Link (Sicherungsschicht)

Physical, die Schicht Eins, beschreibt den Transport der einzelnen Bits durch das Übertragungsmedium, wobei das Medium nicht zur Schicht Eins dazu gezählt wird. Es werden lediglich die elektrischen, mechanischen und funktionalen Schnittstellen zum Medium definiert. Eingesetzte Protokolle unterscheiden sich nach dem eingesetzten Medium (Glasfaser, Kupfer, Funk usw.) und Verfahren (asynchron, synchron, rahmenorientiert usw.).

Data Link dient der Medienzugriffskontrolle mit Adressierung und Vermittlung der Datenpakete an das Endsystem auf Schicht Zwei-Niveau. Mit beinhaltet kann auch eine Datensicherung über Verbindungsabschnitte sein. Eingesetzte Protokolle sind hier typischerweise MAC, PPP oder PPPoE. Letzteres ist vor allem bei ADSL-Anschlüssen in Deutschland im Gebrauch und somit für die Zustellung von IPTV-Inhalten über das Internet erforderlich.

Zu erkennen ist der Informationsfluss jeweils zwischen den einzelnen Domains (Bereichen). Hierbei ist die Adressierung auf Schicht Zwei-Niveau verantwortlich. Die Adressierungsart sieht nur eine Adressierung innerhalb eines Netzwerks vor. Man spricht auch von der MAC-Adressierung. Sie ist Voraussetzung, um Dienste auf höheren Schichten anbieten zu können.

2.3.3.2.2 Network (Netzwerk) und Transport (Transport)

Die Netzwerk-Schicht ermöglicht eine Netz-übergreifende Verbindung auf Schicht Drei-Niveau durch Paketrouting über geeignete Schicht Zwei-Verbindungswege. Damit kann der Informationsfluss netzwerkübergreifend geschehen. Eine Formatierung der Pakete mit einen IP-Header, in der IP-Quell- und Zieladresse hinterlegt sind, sind für die Umsetzung des Paket-Routing's notwendig. Diese Mittel des IP-Protokoll's sind Voraussetzung für den Datenaustausch zwischen einheitlich adressierten Endsystemen eines WAN's.

Aufbauend auf der Schicht Drei setzt die Transport-Schicht auf Schicht Vier auf. Sie dient der Ende-zu-Ende-Verbindung und sichert eine gewisse Übertragungsqualität mittels Fehlerkorrekturverfahren und Schutz vor Paketverlusten. Hierbei spielt das TCP-Protokoll [siehe Glossar] sowie das UDP-Protokoll [siehe Glossar] eine wichtige Rolle. Transportiert werden über TCP im Anwendungsfall von IPTV aber hauptsächlich nur steuerungsrelevante Daten, die für eine Diensteinleitung oder Beendigung gebraucht werden. Lediglich für VoD-Inhalte, die nicht zeitkritisch sind, kann TCP zum Einsatz kommen. Hingegen wird UDP für den eigentlichen Datenfluss verwendet. Durch seine verbindungslose Übertragung werden zeitnahe sowie verzögerungsfreie Zustellungen garantiert. Dies ist vor allem bei Echtzeit-Streaminginhalten für LMB von nutzen.

Die aufgezeigte Delivery Network Domain ist somit für den Verbindungsauf- und Abbau und den Informationsaustausch zwischen dem Endnutzer und seinen Heimnetzwerk und dem Service Provider auf Schicht Drei- und Schicht Vier-Niveau zuständig.

2.3.3.2.3 Session (Sitzung) und Application (Anwendung)

Die Sitzungs-Schicht eröffnet oder schließt die benötigte Verbindung zum Starten oder Beenden einer Anwendung. Dafür setzen Protokolle für die Diensteinleitung, den Transport und der Steuerung auf dieser Schicht auf. Im Fall von DVB-Diensten wird die Schicht MHP [siehe Glossar] genannt.

Der Informationsfluss zwischen dem CP und dem Heimnetzwerk erfolgt auf beiden Schichten direkt und logisch und beinhaltet z.B. eine Rechteverwaltung und eine Absicherung.

2.3.3.3 Das Heimreferenzmodell [14]

Die Architektur eines IPTV-Heimnetzwerks muss verschiedene Szenarien unterstützen. Solche Szenarien können z.B. sein:

- gleichzeitige Verbindung des Heimnetzwerk's zu mehreren Zugangsnetzwerken
- freie Wahl des Dienstanbieter's

---

14 vgl. ETSI TR 102 033 (V1.1.1) „Digital Video Broadcasting (DVB); Architectural framework for the delivery of DVB-services over IP-based networks“

- mehrere Anwender können im selben Heimnetz verschiedene SP wählen usw.

Basierend auf diesen Szenarien lässt sich ein Referenzmodell ableiten. Das Hauptaugenmerk im Referenzmodell liegt vor allem auf der Schnittstelle zwischen dem Heimnetzwerksegment und dem IPTV-Endgerät (siehe Abbildung 2-3). Diese Schnittstelle wird als IPI-1 im Standard deklariert. An diese Schnittstelle werden spezielle Anforderungen bezüglich der Interkonnektivität mit den im Netzwerk befindlichen Komponenten gestellt. DVB-IPI-konforme Mechanismen, wie unter 2.3.3.4 noch erläutert werden, muss diese Schnittstelle unterstützen. Als Endgerät wird in den häufigsten Anwendungsfällen eine Set-Top-Box (siehe 2.3.3.3.1) verwendet.

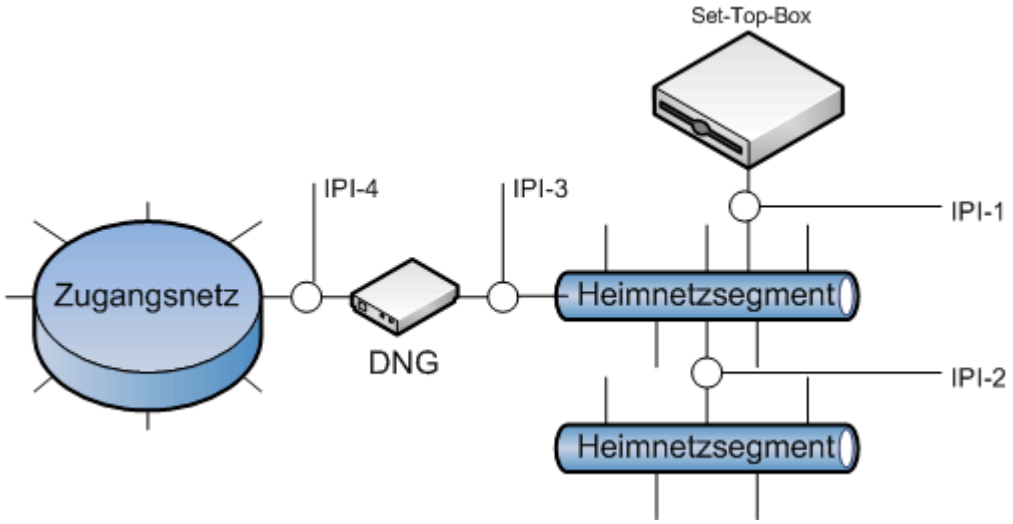

Abbildung 2-3: Elemente im DVB-IPI Heimreferenzmodell

Das DNG ist das Verbindungsglied zwischen ein oder mehreren Netzen beziehungsweise Heimnetzwerksegmenten. Es kann die Verbindung auf mehreren Schichten zwischen dem Zugangsnetz und dem Heimnetz realisieren. Die Kabelverbindung zwischen den Netzen wäre demzufolge eine Schicht Eins-Verbindung und das Routen der Datenpakete eine Schicht Drei-Verbindung.

2.3.3.3.1 Set-Top-Box

Eine sogenannte Set-Top-Box (STB) ist Voraussetzung für den Empfang von IPTV-Diensten und dient als Gateway zum Internet. Sie wird hierbei an den heimischen Fernseher angeschlossen und konvertiert den IP-Datenstrom in ein Fernsehsignal um. Sowohl eine Signalaufbereitung als auch die Software zum Empfang von Inhalten aus IPTV-Kopfstationen sind in der STB enthalten (siehe 2.3.3.4). Die Software ist für die Dekodierung und Aufteilung der Datenströme verantwortlich. Datenströme können Audio-, Daten- oder Videoinformationen enthalten. Oft sind in STB's auch

Entschlüsselungsverfahren implementiert, da von den Dienstanbietern zum Schutz der Inhalte die Datenströme verschlüsselt werden. [15]

2.3.3.4 Transport von Datenströmen über IP-basierte Netze

Für den Transport von DVB-spezifischen Inhalten wie LMB über das IP-Netz muss zunächst eine Umsetzung des Eingangsignal's in ein Ausgangssignal erfolgen, welches mittels geeigneter Protokolle über ein IP-Netz transportiert werden kann. Solche Signalaufbereitungsanlagen sind in sogenannten IP-Kopfstationen verbaut. Üblicherweise werden Satellitensignale oder terrestrische Signale in Form von Transportströmen (TS) in die Kopfstation eingespeist.

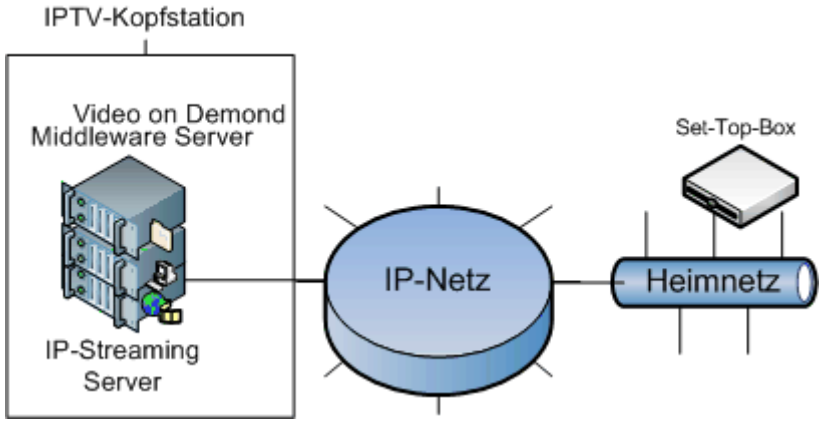

Abbildung 2-4: IPTV-Kopfstation

Der enthaltene TS wird nun innerhalb der Kopfstation zur Aufbereitung an den IP-Streaming Server geschickt. Dort erfolgt eine Einkapselung in ein IP-basiertes Transportprotokoll, meist UDP. Notwendig ist dieser Vorgang für das Routen der Datencontainer durch das IP-Netz. In einem IP-basierten Netz spricht man auch von Datenpaketen. Letztendlich wird der MPEG-2 TS oder H.264/MPEG-4 AVC [siehe Glossar] um Netzwerkprotokolle erweitert (siehe Abbildung 2-5).

Zusätzlich können in einer IP-Kopfstation auch VoD-Server zur Verfügung stehen. Auf den Servern kann Videomaterial dem Kunden bereitgestellt und bei Bedarf von diesen abgerufen werden. Koordiniert wird der Datenaustausch zwischen den Komponenten durch den Middleware [siehe Glossar] Server.

Für die Zustellung von Diensten definiert der Standard bestimmte Mechanismen. Darüber hinaus kann die Zustellung in drei Teilmechanismen untergliedert werden: [16]

---

15 vgl. http://www.itwissen.info/definition/lexikon/Settop-Box-STB-set-top-box.html, Stand: 25.10.2010

16 vgl. ETSI TS 102 034 (V1.4.1) „Digital Video Broadcasting (DVB); Transport of MPEG-2 TS Based DVB Services over IP Based Networks"

Bei einer Diensteinleitung kann nun in Abhängigkeit des ausgewählten Dienstes das aufbereitete Datenpaket mittels Unicast [siehe Glossar] oder Multicast [siehe Glossar] an den Empfänger beziehungsweise die Empfänger adressiert und gesendet werden. Eingeleitet werden Dienste durch Service Discovery and Selection, kurz SD&S. Dabei werden Dienstzustellungsinformationen, Medienformate und Protokolle für die Informationsbeförderung ausgehandelt. Neben der Nutzung des HTTP-Protokoll's [siehe Glossar] für den Transport von SD&S Informationen über Unicast definiert DVB ein neues Protokoll zur Zustellung von XML-Aufzeichnungen über Multicast – das DVB SD&S Transport Protocol (DVBSTP).

Zum Transport der Datenströme wird hauptsächlich das Transportprotokoll UDP verwendet. Sinnvoll ist die Variante bei einem verwalteten Netzwerk. Dort wird meist eine gewisse Garantie hinsichtlich Paketverlusten bei der Paketvermittlung gewährleistet. Allerdings kann ein Datenstrom zusätzlich mit dem höherschichtigen RTP-Protokoll verschachtelt werden. Mit dem Protokoll kann die Zeitbeziehung der Datenströme der Quelle zum Empfänger übertragen werden. Aufgrund der Nähe zur Anwendung wird es der Anwendungsschicht zugeordnet. Somit schlägt RTP die Brücke zwischen dem UDP-Transportprotokoll und der eigentlichen Anwendung.

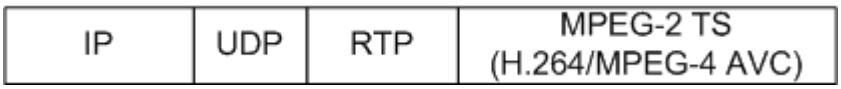

Abbildung 2-5: UDP mit RTP-Verschachtelung

Zur Verbindungssteuerung zwischen Endgerät und IP-Streaming-Server ist abhängig vom ausgewählten Dienst, das IGMP-Protokoll [siehe Glossar] für LMB (Zustellung über Multicast) oder das höherschichtige RTSP-Protokoll [siehe Glossar] für VoD (Zustellung über Unicast) definiert. Da der Konsument bei VoD aktiv am Programmgeschehen teilnimmt, d.h. Auf- und Abbauphasen des Stream's eigens bestimmt, ist die Realisierung über eine Unicast-Adressierung unabdingbar. Hingegen erfolgt die Ausstrahlung von LMB-Datenströmen kontinuierlich und typischerweise über Multicast.

2.3.4 Datenraten

Wie bereits erläutert, gibt es verschiedene Möglichkeiten der Zustellung von IPTV-Inhalten. Bestimmte Dienste erfordern den Zugriff auf ausgewählte Medienserver. Diese Server verursachen in Abhängigkeit ihres Standortes und Anschlusses am Netzwerk einen bestimmten Datenverkehr. Im Client-Server-Modell liefert die sternförmige Verteilung sehr schnell eine Überlastung des Netzes. Einen Ausweg bietet die Peer-to-Peer-Verbindung, bei denen alle beteiligten Geräte Daten konsumieren aber auch an andere Geräte weitergeben können, wenn diese angefordert werden [17].

Bei Unicast steht jedem Benutzer ein eigener Datenstrom zur Verfügung. Dem Nutzer können somit gewisse Freiheiten im Konsum von Übertragungen eingeräumt werden.

17 vgl. http://de.wikipedia.org/wiki/Internet_Protocol_Television, Stand: 13.10.2010

Jedoch steigt die Netzlast bei Unicast in Abhängigkeit mit der Anzahl der Teilnehmer stark an. Beim zweiten Verfahren, dem Multicast, sendet ein Server an mehrere ausgewählte Empfangsstationen gleichzeitig. Dem Teilnehmer kann damit nur ein lineares Programm angeboten werden. Die Anforderungen an das Übertragungsnetz sind jedoch geringer, da die Netzlast nicht mit der Anzahl an Teilnehmern steigt.

Die notwendige Datenrate, um bewegte Bilder übertragen zu können, ist von dem verwendeten Kodierungsverfahren abhängig. Einflussfaktoren sind neben der Bildgröße und Farbtiefe auch die Bildveränderungen und die Anzahl an Audiokanälen. Auch bringen mögliche synchrone Zusatzinformationen eine erhöhte Datenrate mit sich. Dank clientseitiger Puffertechnik sind Übertragungsraten von mehr als 2,5 Mbit/s bei Video-Streams mit hinreichend QoS möglich. Bei SDTV liegt die Datenrate bei 2-6 Mbit/s. Allerdings stellen einige moderne Dienste weitaus höhere Anforderungen an die Bandbreite und der damit verbundenen Datenrate. So werden z.B. bei HDTV Datenraten von 6-16 Mbit/s anfällig. Hierfür ist ein Breitbandanschluss (ADSL3, VDSL4) notwendig. [18]

Bei der Bereitstellung von Triple-Play-Diensten, das heißt Telefon, TV und Internet über einen Anschluss, werden Datenraten von mindestens 15-20 Mbit/s fällig. Dank Kompressionscodierung mit MPEG-2 oder MPEG-4 Formaten kann der Datenratenbedarf zwar gesenkt werden, aber Umschaltzeiten oder Synchronisationszeiten bei einen Senderwechsel steigen dafür an. Die heute übliche Triple-Play-Technik erfordert einen Internetzugang von mindestens 3 Mbit/s [19]. Die nachfolgende Abbildung 2-6 soll einen bildlichen Vergleich der Datenraten einzelner Dienste aufzeigen.

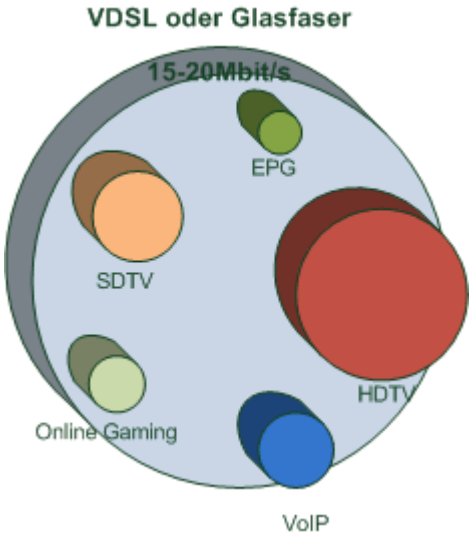

Abbildung 2-6: Erforderliche Datenraten ausgewählter IPTV-Dienste

18 vgl. http://de.wikipedia.org/wiki/Internet_Protocol_Television, Stand: 25.10.2010

19 vgl. http://www.crn.de/datacenter/artikel-5532-4.html, Stand: 25.10.2010

2.4 Technische Grundlagen von Internet-TV

IPTV setzt spezielle Anforderungen an Software-Bausteine und die Ausstrahlung des Inhaltes wird durch Verschlüsselungsverfahren vom Anbieter rechtlich geschützt. Folglich wird zum Wohl des Anwender's eine Standardisierung von benannten Organisationen vorangetrieben. Hingegen erfolgt die Übertragung bei Internet-TV über das „öffentliche“ Datennetz mit seinen bewährten Übertragungsstandards. Somit entfallen die bestimmten Anforderungen aus dem IPTV-Bereich und folglich kommen bekannte Übertragungstechnologien zum Einsatz.

2.4.1 Einleitung

Grundsätzlich erfolgt bei Internet-TV die Übertragung über Video-Streams und stützt sich auf Technologien von Streaming Media. Mit Streaming Media werden empfangene und gleichzeitig wiedergegebene Audio- und Videodaten bezeichnet. Der Vorgang der Datenübertragung wird als Streaming bezeichnet [20]. Ein Streaming-Server wandelt einen Live-Mitschnitt oder eine bereits vorhandene Audio- oder Videodatei mittels eines Encoders in ein Streaming-Format um. Bekannte Streaming-Formate sind MOV, AVI, WMA oder WMV [21]. Hierbei spricht man auch von Containerformaten und diese enthalten Streaming-Codecs. Codecs sind für das Kodieren und Dekodieren von Daten zur Senkung der Datenrate zuständig.

2.4.2 Streaming-Arten und Protokolle

Anhand von Streaming-Protokollen der Anwendungsschicht wie MMS [siehe Glossar], HTTP oder RTP gelangt der Datenstrom vom Streaming Media Server an den Software-Player. Speziell bei On-Demand-Streaming reichen konventionelle Protokolle zum Datentransfer aus [22]. Demzufolge erfolgt die Übertragung mit UDP, während steuerrelevante Daten mittels TCP ausgetauscht werden [23]. Oft wird auch das HTTP-Protokoll für On-Demand-Streaming eingesetzt. Dabei erfolgt die Datenübertragung ausschließlich über das TCP-Protokoll. Für eine lückenlose Übertragung erfolgt eine Zwischenpufferung der Daten. Eine Pausierung der Wiedergabe ist ebenfalls möglich [24].

Hingegen werden bei Live-Streaming die Daten in Echtzeit übertragen. Eingesetzte Protokolle in der Anwendungsschicht für den Transport und Steuerung zeit-sensibler

---

20 vgl. http://de.wikipedia.org/wiki/Streaming_Media, Stand: 12.01.2011

21 vgl. http://de.wikipedia.org/wiki/Streaming-Format, Stand: 12.01.2011

22 vgl. http://de.wikipedia.org/wiki/Streaming-Protokoll, Stand: 12.01.2011

23 vgl. http://de.wikipedia.org/wiki/MMS-Protokoll, Stand: 12.01.2011

24 vgl. http://de.wikipedia.org/wiki/Streaming_Media, Stand: 12.01.2011

Daten sind hier RTP beziehungsweise RTSP. Beide Protokolle finden auch Anwendung im DVB-IPI-Standard.

2.4.3 Streameröffnung

Bei einer Sitzungseröffnung durch ein Präfix in der URL, wie z.B. *mms://*, errichtet der Client mittels TCP eine Verbindung zum Server unter Angabe seiner IP-Adresse und der UDP-Portnummer, über die die Übertragung stattfinden soll. Folglich erzeugt der Server einen UDP-Socket und verbindet ihn mit dem gewünschten Port des Client's. Ein Socket ist ein Kommunikationsendpunkt und dient dem Austausch von Nachrichten zwischen Prozessen.

2.5 Stand der Technik

Schon seit geraumer Zeit werden die Intervalle für technische Neuerungen von Jahr zu Jahr kleiner. Firmen und Konzerne liefern sich erbitterte Kämpfe um die modernsten und technisch anspruchsvollsten Produkte auf dem Markt. Nur zu schnell verliert der Kunde den Überblick über die zur Zeit auf dem Markt befindlichen Produkte und seine Spezifikationen.

Obwohl man annehmen könnte, auch auf dem Sektor von IPTV und Internet-TV so einen Trend erkennen zu können, wird der Fortschritt hier durch bestimmte Kriterien eingeschränkt oder zumindest verlangsamt.

2.5.1 Infrastruktur am Beispiel Deutschland

Ein Blick auf die technischen Anforderungen für den Empfang von IPTV zeigt, dass der Bedarf an Übertragungsrate bei keinen anderen Dienst im Privatkundensegment so hoch ist. Diese Tatsache hat unmittelbare Auswirkung auf den heimischen Internetanschluss.

Laut dem Bundesministerium für Wirtschaft und Technologie besitzen derzeit rund 60% der deutschen Haushalte breitbandfähige Internetanschlüsse. Demnach gilt eine Downstreamrate von mindestens 1 Mbit/s als Breitbandzugang. Bis zum Jahr 2014 sollen für 75% der bundesdeutschen Haushalte Anschlüsse mit Übertragungsraten von bis zu 50 Mbit/s zur Verfügung stehen. Um diese Ziele realisieren zu können, versucht die Bundesregierung Breitbandstrategien umzusetzen. Die Frequenzpolitik nimmt hier eine tragende Rolle ein. Unter anderem soll die Digitale Dividende durch die Digitalisierung der terrestrischen Rundfunkausstrahlung für den Breitbandausbau im ländlichen Raum genutzt werden. Aber auch eine wachstumsfreundliche Regulierung, finanzielle Fördermaßnahmen oder eine Mitnutzung von bereits bestehenden oder neu gebauten Infrastrukturbereichen sollen die Strategie voran treiben. [25]

25 vgl. http://breitbandinitiative.de/news/flaechendeckendes-breitband-in-2010, Stand: 25.10.2010

Doch ein Breitbandanschluss bedeutet nicht gleich den uneingeschränkten Genuss von IPTV nutzen zu können. Eine Mindestübertragungsrate von 3 Mbit/s sind für derzeitige Triple-Play Techniken nötig. Somit sind vielen Bürgern mit ihren verfügbaren Anschlüssen technische Grenzen gesetzt. Gründe liegen hierbei oft an der örtlichen Kabelinfrastruktur. Der sogenannte Flaschenhals liegt in der Verbindung zwischen den OVSt's und den Teilnehmeranschlüssen. Meist sind in ländlichen Gebieten die Kabelwege zwischen den OVSts und den Teilnehmeranschlüssen einige Kilometer lang. Kupferanschlussleitungen mit ADSL stoßen bei Kabelwegen von mehr als 3 Kilometer dämpfungstechnisch an ihre Grenzen. Glasfaserkabel oder VDSL2-Anschlüsse bieten diesbezüglich eine bessere Perspektive. Fiber-to-the-Building oder Fiber-to-the-Home als mögliche Lösungsvarianten sind derzeit im Vormarsch. Dabei reicht das Glasfaserkabel direkt in den Hausanschluss. Von dort kann der Nutzer mit Inhouse-DSLAM's die Umsetzung auf Kupferkabel bewerkstelligen. So können Triple-Play-Dienste pro Teilnehmer mit einer Datenrate von bis zu 100 Mbit/s bereitgestellt werden. [26]

IPTV stellt auch der Technik in den Vermittlungsstellen hohe Anforderungen. TV-Multicast ist mit alten DSLAM's nur bedingt möglich. QoS-Differenzierung und VLAN-Management müssen hier zum Einsatz kommen. [27]

Leider werden diese Techniken vorwiegend im städtischen Bereich vorangetrieben. Zwar wird sich um eine flächendeckende Breitbandversorgung auf dem Land mit der erst kürzlichen Versteigerung der freiwerdenden Frequenzen des analogen Rundfunk's gekümmert. Jedoch bleibt abzuwarten, ob die Nutzung von IPTV-Diensten für diese Bürger in Frage kommen kann.

2.5.2 Technische Geräte

2.5.2.1 Set-Top-Boxen

Entscheidet sich der Nutzer über einen kommerziellen Anbieter IPTV-Inhalte zu beziehen, so wird in der Regel die entsprechende Hardware für den Empfang mit angeboten. Anbieter wie die Telekom oder Alice binden durch eigene Übertragungsmethoden den Kunden an bestimmte Geräte. Dort ist der Empfang allerdings auf eine Methode begrenzt. Oft werden weitere Module wie z.B. für den digitalen terrestrischen Empfang deaktiviert obwohl sie in dem Gerät vorhanden sind.

Verschiedene Hersteller setzen seit geraumer Zeit auf Hybrid-DVB Set-Top-Boxen. Unterstützung finden in den Geräten neben DVB-IPI auch bekannte Standards wie DVB-C, DVB-T oder DVB-S. Dies ist vor allem für Kunden mit Anschlüssen interessant, die weniger Übertragungsrate zur Verfügung haben. Eine Kopplung zwischen unterschiedlichen Übertragungsverfahren ist somit möglich. Zum Beispiel kann der

26 vgl. http://www.crn.de/datacenter/artikel-5532-3.html, Stand: 25.10.2010

27 vgl. http://www.crn.de/datacenter/artikel-5532-4.html, Stand: 25.10.2010

vorhandene Kabel- oder Satelliten-Anschluss für die TV-Übertragung genutzt werden. Gleichzeitig können über den Internetanschluss andere lineare LMB-Dienste oder VoD-Dienste bereitgestellt werden.

Zu den Ausstattungsmerkmalen aktueller Produkte gehören DVR-Funktionalität, HDTV-Receiver (Verstärker), Time-Shift Funktionen, EPG mit Timer-Programmierung, ein Zugang zu Web-Anwendungen und vieles mehr. Integrierte Festplatten aber auch andere Speichermedien geben dem Gerät die Möglichkeit, Daten abzuspeichern. Eine CPU als aktive Hardware dient der Dekodierung von Audio-, Bild- und Videoformaten. Neben HDMI, DVI, SCART und vielen mehr bieten auch optische Anschlussmöglichkeiten für verlustarme Übertragung hinreichend Konnektivität. Die Verbindung zum Internet kann standardmäßig über den Ethernet-Port erfolgen. Optional können an einigen Produkten WLAN-Module nachgerüstet werden. Zum Empfang von Bezahlfernsehen stehen Slots für CA-Module bereit.

2.5.2.2 Internet auf dem Fernseher

Mit Internet@TV bezeichnet der Weltkonzern Samsung seine Technologie, Internet-Inhalte auf dem heimischen Fernseher zu konsumieren. Doch nicht nur Samsung bedient sich mittlerweile des Internet's, auch Philips mit NetTV oder Panasonic mit VieraCast sind Vorreiter in Sachen Internet auf dem Fernseher.

Diese Technologien ermöglichen den Zugriff auf Web-Anwendungen über die Fernbedienung. In Abhängigkeit des Hersteller's erhält der Anwender Zugriff auf Online-Portale wie Youtube, Flickr oder Picasa. Ebenso können soziale Plattformen wie Facebook oder Twitter genutzt werden. Einige Hersteller bieten auch den Zugriff auf andere Webseiten anhand eines integrierten Browsers an. Dabei stehen für die Anpassung von Internetseiten auf TV-Geräte spezielle Standards zur Verfügung. Möglichkeiten zur nachträglichen Erweiterung von Anwendungen bieten herstellereigene App-Stores. [28]

Allerdings stecken die verschiedenen Technologien der Hersteller noch in den Kinderschuhen. Wie aus diversen Erfahrungsberichten zu entnehmen ist, ist die Idee eines integrierten Browsers zwar gut, jedoch durch die umständliche Bedienung über die Fernbedienung noch nicht wirklich alltagstauglich. Das Angebot an Anwendungen aus den App-Stores ist zur Zeit auch noch sehr übersichtlich. Jedoch versprechen die Hersteller, diese kontinuierlich zu füllen.

2.5.2.3 Software-Anwendungen

Richtet man den Blick weg von kommerziellen Endgeräten, so eröffnen sich eine Reihe anderer Möglichkeiten, Inhalte aus dem Internet zu beziehen. Immer mehr kostenlose und in einigen Fällen quell-offene Softwareanwendungen in Form von Media Center bieten

28 vgl. http://www.netzwelt.de/news/82210-internet-fernseher-bieten-tv-hersteller.html, Stand: 22.01.2011

den Zugriff auf Online-Portale oder Mediatheken als Grundfunktion oder auch als Erweiterung an. Damit erschließt sich für den Anwender die nahezu unbegrenzte Vielfalt an Video- oder Bildmaterial aus dem Internet. Auch der Zugang zu sozialen Netzwerken ist bei vielen Anwendungen schon selbstverständlich. Ebenso bieten viele Lösungen die Möglichkeit, aktuelle Wetterdaten von Online-Wetter-Portalen zu beziehen.

Meist kombinieren Anwender die Software-Anwendungen mit energieverbrauchsarmen, leisen aber trotzdem leistungsstarken Media Center-PCs. Neben handelsüblichen Lösungen kann sich der Anwender mit ein wenig Hintergrundwissen die Komponenten dafür selbst zusammenstellen. Anstelle von Media Center-PC wird oft der Begriff Home Theatre Personal Computer verwendet.

2.5.3 Zukunftsaussichten

Die Ausgangssituation für IPTV in Deutschland war mit Beginn der Einführung denkbar schlecht. Eine weitreichende Auswahl an Free-TV-Programmen über Kabel, Satellit oder Antenne war schon vorhanden. Außerdem fehlte es vielerorts an der benötigten Infrastruktur für IPTV.

Mit der stetigen Verbesserung von Infrastruktur und Breitbandausbau wird jedoch die Anzahl der Abonnenten von IPTV steigen. Prognosen und Schätzungen aus Studien prophezeien, dass bis 2013 nahezu 2,5 Millionen Kunden das Angebot nutzen werden [29]. Folglich werden auch immer mehr Anbieter von IPTV-Angeboten auf den Markt drängen. Somit wird der marktwirtschaftliche Wettbewerb angeregt. Dies ist wiederum für den Kunden von Vorteil.

Doch nicht nur kommerzielle IPTV-Anbieter werden vom Breitbandausbau profitieren. Der Trend von der herkömmlichen Videothek zur Online-Videothek ist vorauszusehen. Daher werden immer mehr Anbieter solcher Online-Portale oder Mediatheken auf den Markt drängen.

29 vgl. http://www.iptv-anbieter.info/, Stand: 22.10.2010

3 Präzisierung der Aufgabenstellung

In diesem Kapitel erfolgt die Darstellung des Ist-Zustandes und die Präzisierung der Aufgabenstellung. Dazu erfolgen einleitende Worte zum Home Theatre Personal Computer und eine anschließende detaillierte Auseinandersetzung mit dem Xbox Media Center. Es werden Anforderungen, Leistungsmerkmale und Funktionen beschrieben. Des Weiteren erfolgt eine Erläuterung der Programmierschnittstelle im Xbox Media Center.

3.1 Home Theatre Personal Computer [30]

Als Home Theatre Personal Computer (HTPC) wird ein auf PC-Komponenten basierendes Gerät verstanden. Es soll durch seinen modularen Aufbau herkömmliche Hi-Fi-Geräte als eine Art Medien Center-PC ablösen. Eine weit verbreitete Bezeichnung ist nebenbei noch Media Center-PC, wobei im deutschsprachigen Raum der Begriff Media Center sehr gebräuchlich ist.

Zu den Aufgaben eines HTPC gehören:

- Abspielen von Filmen und Videos von CD, DVD, HD-DVD, Blu-ray u.a. Formate
- Abspielen von Musik von CD, MP3 u.a. Formate
- Betrachten von Bildern
- Abspielen von analogen bzw. digitalen Fernsehens (DVB-C/S/T)
- Aufnahme von TV-Programmen über DVR

Der modulare Aufbau ermöglicht eine hohe Flexibilität und dadurch eine individuelle Anpassung an eigene Wünsche. Leicht können so weitere Funktionen nachgerüstet werden, wie zum Beispiel:

- eine Spielkonsole
- Internet-TV (VoD-Dienste, ARD/ZDF-Mediathek, YouTube und vieles mehr)
- Media-Server

Aufgrund dieser Vielfältigkeit kann ein HTPC alt bekannte Heimgeräte wie DVD-Player, CD-Player, DVB-Receiver und Videorekorder ersetzen. Des Weiteren kann er als Verteiler anderer im Haus befindlichen Clients für multimediale Inhalte dienen.

---

30 vgl. http://de.wikipedia.org/wiki/Home_Theater_Personal_Computer, Stand: 28.10.2010

Bei einer Aufrüstung zum internetfähigen Media Center kann nach Definition allerdings nicht von IPTV gesprochen werden, vielmehr von Internet-TV (siehe 2.1). Folglich kann es dadurch zu Bild- und Tonqualitätseinbußen führen. Allerdings stößt diese Form des Internet-TV durch seine meist kostenfreie Nutzung bei vielen Nutzern auf reges Interesse.

3.1.1 Allgemeine Hard- und Softwareanforderungen

3.1.1.1 Hardware [31]

In der Planung und Aufbau eines HTPC sollten in Vorbetracht einige Faktoren berücksichtigt werden. Hardwareanforderungen, Stromverbrauch und Geräuschentwicklung sind wichtige Einflussfaktoren bei der Anschaffung eines Gerätes.

Beim Gehäuse sind dem Nutzer praktisch keine Grenzen gesetzt. Der Markt bietet von miniATX-Gehäusen bis Full-Size-Gehäusen alles an. Speziell für den HTPC-Gebrauch entworfene Gehäuse oder umgebaute DVD-Player-Gehäuse sind ebenso Möglichkeiten, den HTPC unterzubringen.

Wichtig ist die richtige Wahl des Prozessor's. Um hochauflösende Inhalte oder Blu-ray zu schauen, sind leistungsstarke Kerne für die Dekodierung nötig. Mit der Rechenleistung von Mehrkern-Prozessoren oder Atom-Prozessoren lassen sich die Inhalte flüssig darstellen. Vor allem liegt der Stromverbrauch bei den neuen Atom-Prozessoren im Vergleich sehr niedrig. Die daraus resultierende geringere Wärmeabgabe ist wiederum mit der Geräuschentwicklung durch die Lüfter verknüpft. So lassen sich unter Umständen Lüfter mit wenig Drehzahl oder Passiv gekühlte Systeme verbauen.

Bei der Wahl des Mainboard's bedarf es auch einer gründlichen Planung. Moderne Mainboards können eine integrierte GPU enthalten. Dies hat den Vorteil, dass für eine dezidierte GPU kein weiterer Platz verbraucht wird. Oftmals ist der Stromverbrauch einer integrierten GPU auch wesentlich geringer. Auch hier sollte auf die Rechenleistung der GPU Rücksicht genommen werden. Software verschiedener Anbieter oder Entwickler unterstützen die Hardwarebeschleunigung von HD-Inhalten darüber. Außerdem sollte das Mainboard über ausreichend Anschlüsse verfügen, wie z.B. HDMI, DVI, eSATA oder USB.

Für den Genuss von digitalen Mehrkanalton aus modernen HD-Tonformaten ist wiederum eine besondere Auswahl der Soundkarte zu treffen. Zwar bieten einige Hersteller Onboard-Sound auf Mainboards an, jedoch sind die Hard- und Software-Voraussetzungen nicht immer gegeben. Auf die von Blu-ray-Discs vorhandenen Soundformate Dolby Digital True HD oder DTS-HD Master Audio werden zurzeit von einer handvoll dezidierten Soundkarten unterstützt.

---

31 vgl. http://3dfusion.de/artikel/preview/Home_Theater_Personal_Computer/1_Einleitung_Die_passende_Hardware/, Stand: 28.10.2010

3.1.1.2 Software

Nach der Aufrüstung eines HTPC mit geeigneter Hardware muss sich der Nutzer für ein passendes Media Center entscheiden. Dabei reicht die Auswahl von integrierten Media Center in Betriebssystemen über eigenständige Media Center-Betriebssysteme bis zu reinen Media Center-Anwendungen.

Der Vorteil von Media Center besteht in der Steuerung. Hier kann der Nutzer alles aus einer Oberfläche steuern. Bilder, Filme, Musik oder auch das aktuelle Wetter können aus Menüpunkten aufgerufen werden. Eine IR-Fernbedienung ermöglicht bei vielen Anwendungen eine Fernsteuerung. Jedoch fehlt in den meisten Fällen die Unterstützung von Blu-ray-Discs. Ohne zusätzliche Plugininstallationen ist hier keine Abhilfe zu schaffen.

Individuelle Zusammenstellungen sind mit einzelnen Softwarekomponenten möglich. Nachteilig ist dabei die unkomfortable Steuerung. Der Nutzer muss zwischen den einzelnen Anwendungen ständig hin und her springen. Allerdings bieten einige Softwareplayer eine Unterstützung des Blu-ray Format's an. Diesen Dienst lassen sich allerdings die Anbieter zum Teil teuer bezahlen.

3.1.2 Konzepte für den Aufbau

Zwei grundlegende Konzepte sind für den Aufbau eines Media Center in Erwägung zu ziehen. Zum Einen die Option als Multifunktionsgerät, zum Anderen als verteilte Anwendung auf Grundlage des Client-Server-Modell's. In Abhängigkeit vom geplanten Einsatz, räumlicher Beschaffenheit oder technischer Infrastruktur muss der Nutzer die bestmögliche Lösung für seine Bedürfnisse treffen.

3.1.2.1 Einsatz als Multifunktionsgerät – die All-in-One-Lösung [32, 33]

Beim Aufbau als All-in-One-Lösung werden alle benötigten Komponenten mit deren Funktionen in einem Gehäuse vereint. Nach dem Booten (Hochfahren) startet hier in den meisten Fällen direkt eine Software – das Frontend. Jedoch kann auch die Software Backend und Frontend vereinen. Bildlich betrachtet sitzen Backend und Frontend in diesem Fall direkt hintereinander. Wobei das Backend im Vergleich zum Frontend Hardware-näher anzusiedeln ist. Das Frontend verarbeitet und optimiert die Daten für die Darstellung auf dem Ausgabegerät und bedient sich dabei von Funktionen aus Backendprogrammen. Auch kann es für den Empfang von Fernsehsignalen und anderen Informationen, wie der Wettervorschau, dienen.

Sicherlich liegt der Vorteil in diesem Konzept im günstigeren Anschaffungspreis und bei dem geringeren Platzbedarf gegenüber Einzelgeräten. Der geringere Energieverbrauch

---

32 vgl. http://de.wikipedia.org/wiki/Home_Theater_Personal_Computer, Stand: 29.10.2010

33 vgl. http://de.wikipedia.org/wiki/Multifunktionsger%C3%A4t, Stand: 29.10.2010

stellt in der heutigen Zeit auch ein interessantes Kriterium dar. Weitere Vorteile liegen in der zentralen Konfiguration und dem einheitlichen Bedienkonzept. Kompromisse müssen allerdings oft in qualitativen und funktionalen Bereichen eingegangen werden. Da das Gerät viele Elemente vereinen muss, fehlen durch die mangelnde Spezifizierung mitunter praktische Geräteausstattungen oder Funktionen.

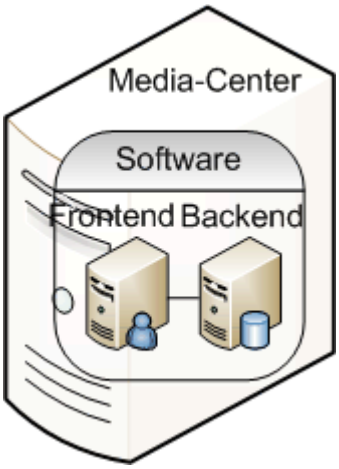

Abbildung 3-1: All-in-One-Lösung

3.1.2.2 Aufbau nach dem Client-Server-Modell [34]

Charakteristisch für dieses Konzept ist der zentrale Server im Netzwerk. Dieser Server verwaltet mit dem Backend alle auf dem Speichermedium befindlichen Multimediadateien und streamt die Daten bei Bedarf an einen oder mehrere Clients. Als Server dient meist ein leistungsstarker HTPC. Hingegen können als Clients einfache Abspielgeräte mit passenden Frontend den Transport zum Anzeigegerät bewerkstelligen. In vielen Fällen besitzt der Server mehrere TV-Tuner. Dabei dient dann der Client als Set-Top-Box. Mit Hilfe einer clientseitigen Netzwerkschnittstelle kann der Anwender über ein Netzwerk auf die Datenressourcen im Server zurückgreifen.

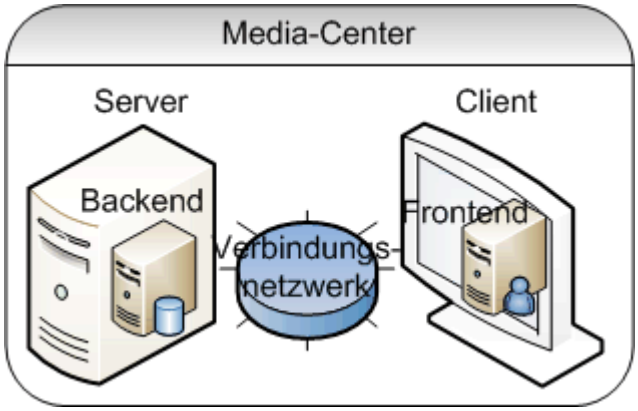

Abbildung 3-2: Aufbau als verteilte Anwendung

34 vgl. http://de.wikipedia.org/wiki/Home_Theater_Personal_Computer, Stand: 01.11.2010

Wesentlicher Vorteil des Konzeptes ist die zentrale Lage der im Netzwerk befindlichen Daten. Dadurch ist nur eine einmalige Einspeisung der Daten nötig. Demzufolge konzentriert sich die Verwaltung auf nur einen Punkt. Somit entfällt relativ teure Gerätetechnik auf der Clientseite.

Entscheidet sich der Nutzer für ein Client-Komplettsystem, muss mit Einschränkungen in unterstützten Dateiformaten gerechnet werden. Für die Dekodierung sorgen Geräte-interne Chips. Diese können oft nur bestimmte Formate dekodieren. Ein Aufrüsten solcher Geräte ist in den meisten Fällen nicht möglich. Abhilfe schafft ein teurer Client auf PC-Basis. Durch den meist tagelangen Betrieb des Server's erhöht sich folgerichtig auch der Strombedarf.

3.1.3 Media Center-Betriebssysteme

Unter Media Center-Betriebssystemen wird ein speziell auf HTPC angepasstes Betriebssystem verstanden. Gegenüber reinen Betriebssystemen enthalten Media Center-Betriebssysteme zusätzliche Softwarekomponenten. Diese ermöglichen mit der passenden Hardware z.B. einen Fernsehempfang, Sendungen aufzuzeichnen, DVDs abspielen oder Bilder entsprechend wiederzugeben. Gerade bei Linux setzen Media Center-Betriebssysteme oft auf abgespeckten Standardbetriebssystemen mit einer minimalistischen Bedienoberfläche auf.

3.1.4 Media Center-Anwendungen

Im Gegensatz zu Media Center-Betriebssystemen lassen sich die Anwendungen beziehungsweise das Frontend aus dem installierten Betriebssystem heraus starten. Interessant ist die Alternative für Anwender, die neben einer HTPC-Software einen voll funktionsfähigen PC mit Office-Anwendungen oder ähnlichen betreiben wollen.

3.2 Vorstellung des Xbox Media Center's

Derzeit stehen eine Vielzahl von Media Center dem Nutzer zur Auswahl bereit. Mit umfangreichen Funktionen, einer besseren Bedienbarkeit, unterstützten Dateiformaten und vielen mehr versuchen sich die Anbieter und Entwickler voneinander abzuheben. Aber auch grundlegende Entscheidungskriterien müssen im Vorfeld abgewogen werden. In Abhängigkeit vom Betriebssystem, dem Einsatz des System's oder der verbauten Hardware sind Entscheidungen zu treffen.

Das Xbox Media Center, kurz XBMC, ist ein unter der GPL [siehe Glossar] -Bestimmung frei verfügbares, plattformunabhängiges und quell-offenes Media Center und dient als Netzknoten digitaler Medien im Heimnetz. Es bietet für den Anwender eine Reihe an Optionen an, Mediendateien abzuspielen, zu verwalten oder das Media Center mit Hilfe von Add-ons zu erweitern. Außerdem bietet es die Möglichkeit, Audio- und Video-Streams aus dem Internet abzuspielen. Ursprünglich wurde die Software für die Xbox entwickelt

und so für einen Ressourcen-schonenden Umgang mit der verbauten Hardware programmiert.

Das Projekt wurde 2003 ins Leben gerufen und seitdem von mehr als 50 Softwareentwicklern und 100 Übersetzern weiterentwickelt beziehungsweise in mehr als 30 Sprachen übersetzt. [35]

XBMC dient als ideale Lösung für das Heimkino. Dafür sorgen eine Vielzahl unterstützter Fernbedienungen zur Steuerung und eine ausgereifte Schnittstelle mit der dahinter liegenden XBMC Skinning Engine.

Nahezu alle derzeit gängigen Audio- und Videoformate kann das Media Center wiedergeben. Dank einer Reihe von unterstützten Netzwerk- und Internetprotokollen ist das Media Center für den Einsatz im Heimnetzwerk oder Internet konzipiert. Multimediainhalte können somit von allen Punkten aus im Heimnetz gestreamt beziehungsweise direkt aus dem Internet bezogen werden. Daten von optischen Medien können ebenso wie von Festplatten oder externen Speichermedien abgespielt werden. Zusätzliche Bibliotheksfunktionen erleichtern die Verwaltung der Multimediadaten.

Hinzukommende Playlist- und Slideshow-Funktionen, Audio-Visualisierungen sowie RSS-Feeds [siehe Glossar] und eine integrierte Wettervorschau lassen XBMC zu einer vollwertigen Multimedia-Jukebox mit interaktiven Elementen verwandeln. [36]

Abbildung 3-3: Hauptmenü des XBMC Media Center

35 vgl. http://wiki.xbmc.org/index.php?title=Introduction, Stand: 03.11.2010

36 vgl. http://xbmc.org/about/, Stand: 03.11.2010

3.2.1 Hardwareanforderungen

XBMC wurde von den Entwicklern dahingehend ausgelegt, auf einer relativ leistungsschwachen Hardware durch Ressourcen-schonenden Umgang zu laufen. Lediglich wird für die Darstellung der Benutzeroberfläche eine leistungsgerechte 3D-Grafikkarte benötigt. Außerdem stellt das Media Center eine Videobeschleunigung über die Grafikkarte bereit, um ein Abspielen von HD-Videomaterial (720p, 1080p) zu gewährleisten.

Empfohlene Mindestanforderungen der Hardwarekonfiguration:

- x86 Intel/AMD Prozessoren: Intel Pentium 4, Intel Pentium M, AMD Athlon XP/64, AMD Opteron, für die Dekodierung von h.264 kodierten Videos in 720p/1080p wird ein Dual-Core Prozessor empfohlen
- ATI/AMD, Intel, NVIDIA Grafikkarte: ATI Radeon R700 (HD 4000), Intel GMA X4500HD (G45), NVIDIA Geforce 8-Serie
- Audio für Linux mit ALSA-Unterstützung, unter Windows mit DirectSound-Unterstützung

3.2.2 Leistungsmerkmale

Durch die Unterstützung der derzeit aktuellsten und gebräuchlichsten Audio- und Videoformate, der Möglichkeit der Datenverwaltung und die Erweiterung des Media Center's zu einer interaktiven Schaltzentrale bietet XBMC dem Nutzer umfangreiche Leistungen an.

3.2.2.1 Formate [37]

Eine Reihe von Video- und Containerformaten sowie Netzwerkprotokollen werden durch XBMC unterstützt. Die gängigsten Formate und Protokolle im Überblick:

- Physische Geräte und Medienformate: CD, DVD, AudioCD, Video-CD (VCD/SVCD/XVCD), USB Flash Drive, Festplatten
- Netzwerkprotokolle: UPnP (DLNA), SMB/Samba/CIFS, FTP, HTTP, DAAP für die Verbindung mit Apple's iTunes
- Containerformate: AVI, MPEG, WMV, ASF, FLV, MKV, QuickTime, MP4
- Videoformate: MPEG-1, MPEG-2, MPEG-4 SP und ASP, H.264/MPEG-4 AVC, WMV

37 vgl. http://de.wikipedia.org/wiki/XBMC_Media_Center, Stand: 03.11.2010

- Audioformate: MIDI, WAV/WAVE, MP2, MP3, AAC, AC3, DTS, FLAC
- Bildformate: BMP, JPEG, GIF, PNG
- Metadaten wie IDv3-Tag bei MP3s usw.

3.2.2.2 Bibliotheksfunktion

XBMC bietet dem Nutzer des Weiteren die Möglichkeit, seine Mediendateien im Media Center in einer Bibliothek zu organisieren. Dabei nutzt das Konzept die starke Anbindung an das Internet, bei dem mit Hilfe eines sogenannten Scraper [siehe Glossar] Metadaten [siehe Glossar], beispielsweise über Filme (Besetzung, Handlung, Filmposter, Filmkritiken) aus Online-Filmdatenbanken aus dem Internet bezogen werden können. Dadurch wird eine Einordnung der Filme nach Kategorien ermöglicht. Der Nutzer hat nun die Möglichkeit, seine Filme nach Darstellern, Regisseuren oder anderen Gesichtspunkten auszuwählen. Hinter der Bibliothek verbirgt sich eine Datenbank, in der die Inhalte abgelegt werden. Dafür verwendet XBMC das Datenbanksystem SQLite3. Während der Laufzeit des Media Center's wird die Datenbank automatisch gewartet und bei Bedarf aktualisiert.

Die nachfolgende Abbildung 3-4 zeigt die Möglichkeit, Inhalte aus einen ausgewählten Medienordner in die virtuelle Bibliothek aufzunehmen. Unter dem Menüpunkt *Videos* kann der Nutzer nach Auswahl des einbezogenen Filmordner's unter der Auswahl *Inhalt festlegen* zwischen drei Inhalten (*Filme*, *Musikvideos*, *Serien*) wählen. Passende Scraper werden automatisch hinzugefügt. Nun werden nach weiteren Einstellungsmöglichkeiten die Inhalte gescannt und automatisch in die Datenbank eingetragen.

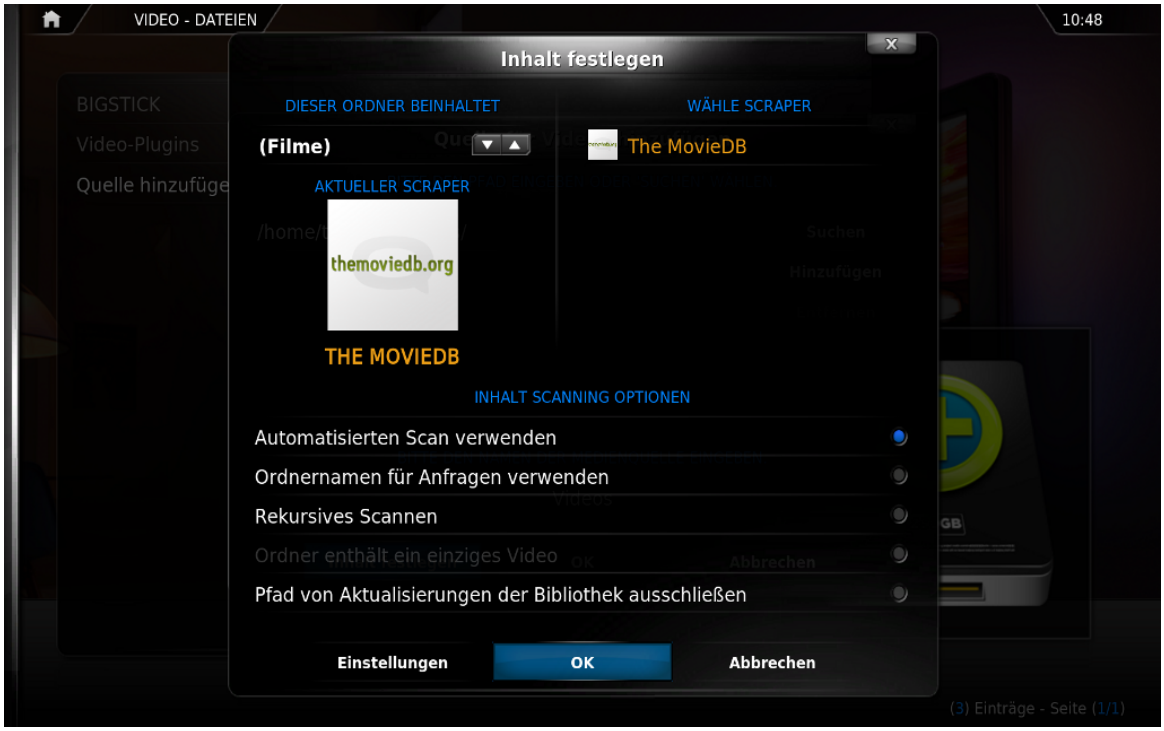

Abbildung 3-4: Integrierte Bibliotheksfunktion in XBMC

3.2.2.3 Add-on's (Erweiterungen)

Mit Python-basierten Add-ons kann XBMC an individuelle Wünsche angepasst werden. Diese Add-ons werden direkt über das Benutzerinterface installiert und verwendet. So können z.B. Internet-basierte Inhalte (YouTube-Videos o.ä.) einfach in das Media Center eingebunden und über darauf zugeschnittene Funktionen (Browser oder ähnliches) angezeigt werden.

Auch Live TV über DVB-S, DVB-T- oder DVB-C kann mit Add-ons in XBMC eingebunden werden. Dies geschieht über die Video Disc Recorder-Software (VDR-Software). VDR-Software ist frei, nicht-kommerziell und dient dem Empfang und der Aufnahme von digitalen Fernsehen. Dafür werden von Entwicklerteams angepasste Versionen mit PVR-Unterstützung angeboten. Ausgestrahlte Inhalte können so auf lokalen Speichermedien aufgenommen und gespeichert werden. Der Unterschied zu einen normalen Videorekorder besteht darin, dass der PVR das Videosignal komprimiert und anschließend digital abspeichert. Befindet sich der PVR in einem Netzwerk, so hat der Nutzer die Möglichkeit, die Inhalte mit speziellen Plugins auf andere Rechner zu streamen.

So wird das Media Center zu einer multifunktionalen Einheit und braucht sich hinter handelsüblichen Set-Top-Boxen nicht zu verstecken.

3.2.2.4 Fernsteuerung und UPnP

Mit dem integrierten Web-Interface wird entfernten Rechnern über den Webbrowser ein Fern-Zugriff (Remote-Zugriff) auf ausgewählte Funktionen in XBMC gewährt. Jedoch müssen sich das Media Center und der Remote-PC im gleichen Netzwerk befinden. Um die Funktion nutzen zu können, muss unter *System → Netzwerk* in XBMC der Webserver aktiviert werden. Standardmäßig „hört“ der Server auf Port 8080. Benutzername und Kennwort können optional vergeben werden (siehe Abbildung 3-7). Sollte hier die Standardeinstellung beibehalten werden, findet keine Abfrage bei der Verbindung zum Server statt.

Abbildung 3-5: Remote-Zugriff über Web-Server

(Bildquelle: http://wiki.xbmc.org/index.php?title=File:Webserver.jpg, Stand: 08.11.2010)

Erreichbar ist das Media Center dann unter *http:///<ip wo sich xbmc befindet>:8080/*. Übermittelt werden die Befehle mit Hilfe der HTTP-API. In ihr sind Funktionen und Methoden für den Remote-Zugriff auf die Funktionalität von XBMC implementiert.

Über das HTTP-API können ausgewählte mobile Endgeräte im WLAN ebenfalls die Steuerung übernehmen. Apple's IPhone, IPad, das IPod touch oder Android-Geräte werden mit Apps um diese Funktion erweitert. Je nach Funktionsumfang sind diese Programme kostenpflichtig beziehungsweise kostenfrei im Appstore oder Android Store erhältlich.

Abbildung 3-6: XBMC Remote-App auf dem Apple IPhone

Darüber hinaus unterstützt das Media Center UPnP [siehe Glossar]-Funktionalität. Über einen integrierten UPnP-Server wird der Zugriff auf freigegebene Ordner und deren Inhalten durch einen im Netzwerk befindlichen UPnP-Client ermöglicht. So kann der

integrierte Server in XBMC seine eingebundenen Inhalte an die Clients bei Bedarf verteilen. Gleichzeitig kann XBMC aber auch als Client fungieren.

Unter dem Menüpunkt *System → Netzwerk* kann sowohl der Server als auch der Client (*UPnP Renderer aktivieren*) aktiviert werden. Die folgende Abbildung veranschaulicht diesen Sachverhalt.

Abbildung 3-7: Aktivierung des UPnP-Server, UPnP-Client und Webserver

Der Zugriff auf freigegebene Inhalte kann nun auf Seiten des Client's mittels *Quelle hinzufügen* und der entsprechenden Auswahl *UPnP Device* realisiert werden. Danach erscheint im jeweiligen Menüpunkt *Audio*, *Video*, *Bilder* oder *Programme* der Link zur Freigabe.

Seit Ende 2007 sind die UPnP-Funktionen in XBMC kompatibel mit DLNA [siehe Glossar] -zertifizierten Geräten.

3.2.2.5 Interaktive Dienste

Natürlich dürfen typische Web 2.0 [siehe Glossar] -Anwendungen in XBMC nicht fehlen. Über den integrierten RSS-Reader können Nachrichten oder Informationen, sogenannte RSS-Feeds, von ausgewählten Webseiten bezogen werden. Eingebunden und in XBMC zur Anzeige gebracht werden die Feeds mittels der RSSFeeds.xml-Datei im Installationsordner unter *userdata*. Doch auch hier wird dem Nutzer das Hinzufügen von eigenen Feeds erleichtert. Spezielle RSS-Editor-Plugins erlauben ein einfaches Editieren der Datei über die Benutzerschnittstelle von XBMC.

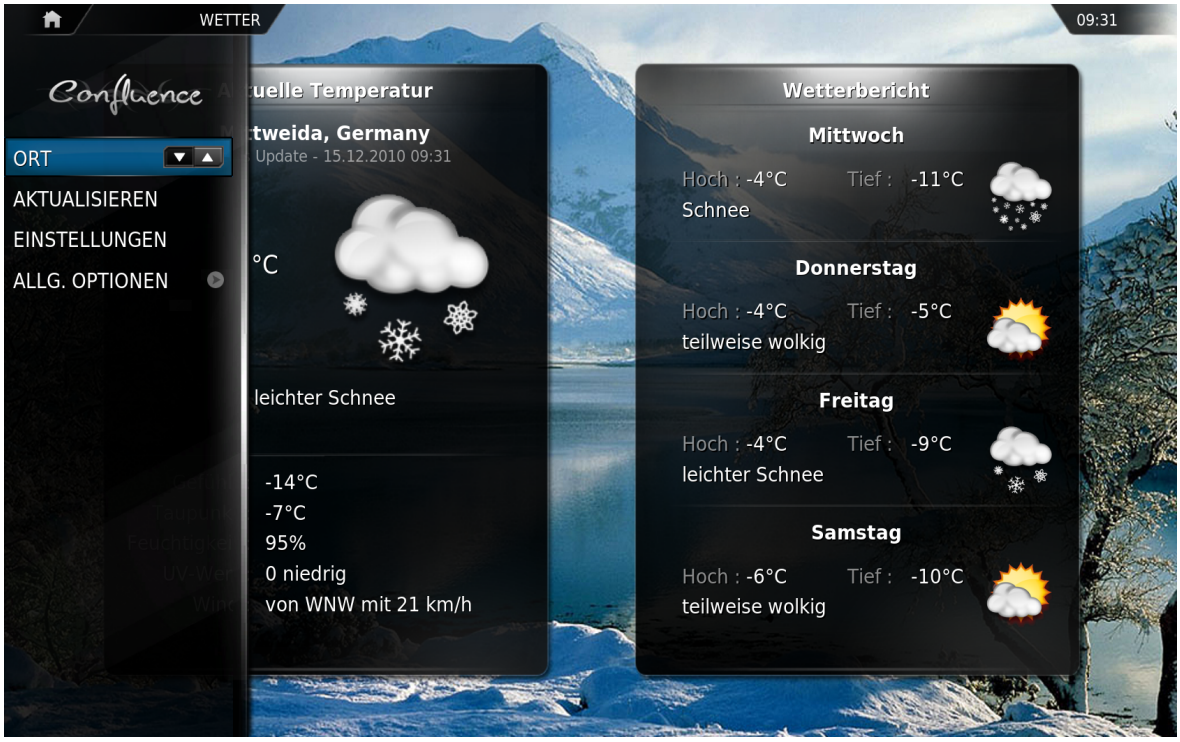

Abbildung 3-8: integrierte Wetterdienst in XBMC

Ein weiterer interaktiver Dienst bietet XBMC unter dem Menüpunkt *Wetter* (Abbildung 3-8). Darunter kann für eine bestimmte Region oder Stadt die Wettervorhersage abgerufen werden. Die Wetterdaten werden von *weather.com* bezogen. Prognosen von drei verschiedenen Standorten kann der Nutzer über die Menüleiste abfragen. Darüber hinaus schaltet XBMC regionsabhängig zwischen Celsius und Fahrenheit umher.

In der Abbildung ist die Menüleiste unter dem Hauptmenüpunkt *Wetter* zu sehen. Neben der Auswahl des Ortes können die Wetterdaten manuell aktualisiert werden. Unter *Einstellungen* kann der Nutzer seine Regionen und Orte individuell auswählen.

3.2.3 Die Programmierschnittstelle – XBMC Python Engine

Um das Media Center mit zusätzlichen Funktionen auszustatten, wird von den Entwicklern ein integrierter Python-Interpreter mitgeliefert. Damit können Add-ons in Form von Plugins oder Skripten leicht programmiert und über die Schnittstelle mit der Oberfläche von XBMC verknüpft werden. Als Grundlage zur Erweiterung des XBMC dienen neben der unterstützten Python-Engine das XBMC-GUI Interface mit seiner integrierten Add-on-Unterstützung. [38]

38 vgl. http://wiki.xbmc.org/index.php?title=Python_Development, Stand: 09.11.2010

3.2.3.1 Eine kurze Einführung in Python unter Linux [39]

Python ist eine höhere universelle Programmiersprache und wurde Anfang der 1990er Jahre als Nachfolger der Programmierlehrsprache ABC entwickelt. Hauptsächlich wird Python als Skriptsprache eingesetzt und unterstützt vorwiegend objektorientierte aber auch aspektorientierte und funktionale Programmierparadigmen. Hauptaugenmerk der Entwicklung lag auf der Einfachheit und Übersichtlichkeit. So kommt Python mit relativ wenig Schlüsselwörtern und einer reduzierten Syntax aus. Dennoch kann Python durch zusätzliches Einbinden von Modulen zu einer mächtigen Programmiersprache geformt werden. Benötigte Software zur Programmentwicklung ist durch die nicht-kommerzielle Programmiersprache kostenlos verfügbar. [40]

3.2.3.1.1 Was heißt Objektorientiert?

„In der objektorientierten Sichtweise stellt man sich die Welt als Systemen von Objekten vor, die untereinander Botschaften austauschen“. (Weigand, Objektorientierte Programmierung mit Python, 3., aktualisierte Auflage 2006, S. 32)

Bei der objektorientierten Programmierung wird ein großes System in mehrere kleine Teilsysteme mit Hilfe von Verfahren untergliedert. Kleinere Teile lassen sich einfacher programmieren und die Fehlerauftrittswahrscheinlichkeit minimiert sich auf ein wesentliches.

3.2.3.1.2 Zusammenhang Objekt – Attribut – Klasse - Methode

Ein Objekt ist ein Exemplar eines Datentyps oder einer bestimmten Klasse. So wird ein Objekt auch als Instanz einer Klasse bezeichnet. Jedes Objekt wird mit Eigenschaften (Attribute) vollständig beschrieben.

Attribute sind Merkmale eines ganz bestimmten Objektes und werden durch einen Typ vereinbart. Zum Beispiel sind Attribute einer Set-Top-Box Dekodierung von Videoinhalten, Aufzeichnen empfangener Daten usw.

Klassen werden über die Gesamtheit von Attributen gleicher Objekte definiert.

Mit sogenannten Methoden können Objekte Operationen ausführen. Somit wird das Verhalten eines Objektes von seinen Methoden bestimmt.

3.2.3.1.3 Der Python-Interpreter

Grundsätzlich liegt die Aufgabe eines Interpreters in der Übersetzung eines Programmtextes zu einen ausführbaren Programm. Dabei liest der Interpreter im Gegensatz zum Compiler den Programmtext zeilenweise ein und führt direkt jede

39 vgl. Weigand, Objektorientierte Programmierung mit Python, 3., aktualisierte Auflage 2006

40 vgl. http://de.wikipedia.org/wiki/Python_%28Programmiersprache%29, Stand: 03.11.2010

Anweisung aus. Demnach sind zu allen Programmen, geschrieben in der jeweiligen Sprache, ein passender Interpreter erforderlich.

Eine Besonderheit des Python-Interpreter's ist der Aufruf im interaktiven Modus. Hier kann man einzelne Zeilen Programmtext eingeben und die Wirkung sofort beobachten (siehe Abbildung 3-9).

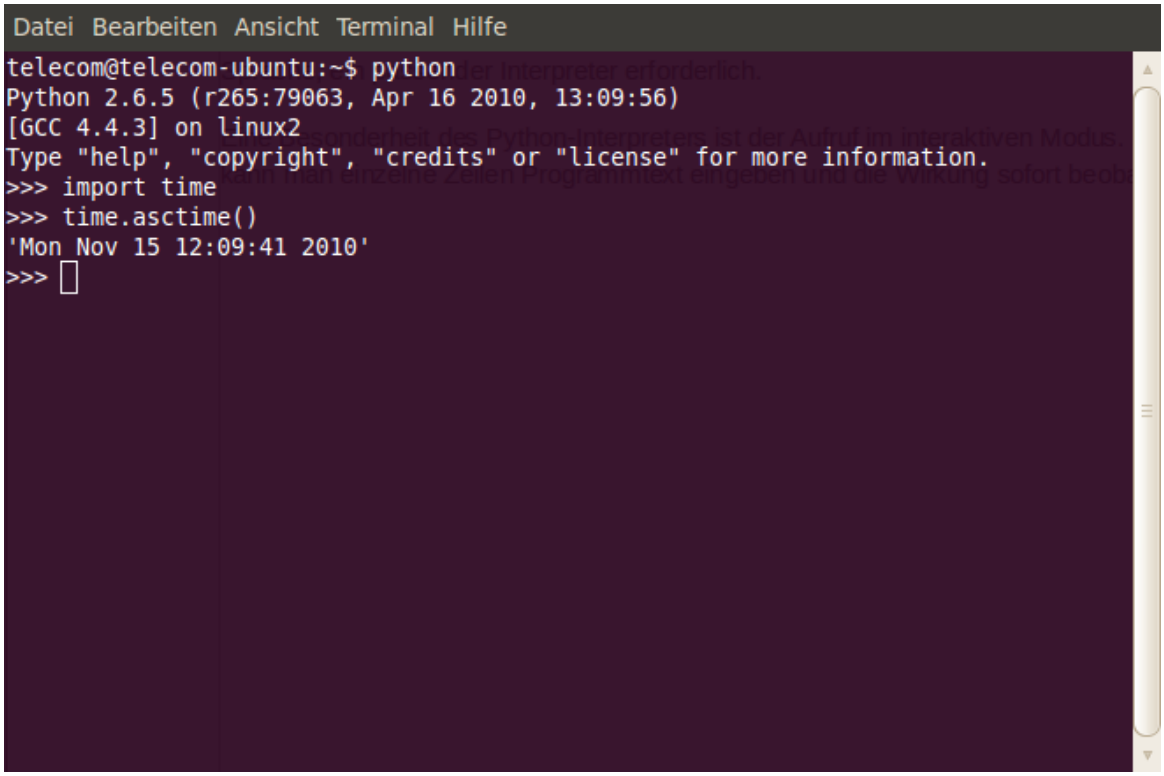

Abbildung 3-9: Interaktiver Modus des Python-Interpreter

Mit Hilfe des Befehl's *python* wird in der Linux-Shell der interaktive Modus gestartet. Eingegebener Programmtext wird sofort übersetzt und ausgegeben. In der oben aufgeführten Abbildung wird nach dem Import des Moduls *time* die Methode *asctime()* auf das Modul angewendet. Folglich erscheint das aktuelle Datum mit Uhrzeit in der Anzeige. Die Syntax wird im Abschnitt noch genauer erläutert.

3.2.3.1.3.1 Module

Module besitzen Definitionen von Klassen, Funktionen und Konstanten. Bevor ihre Objekte genutzt werden können, müssen Module importiert werden.

3.2.3.1.3.2 Schlüsselwörter

Schlüsselwörter sind in der Programmiersprache bereits mit einer Bedeutung belegt. Folgerichtig dürfen diese Wörter nicht als Bezeichner (Zeichenketten) verwendet werden. Wichtige Schlüsselwörter in Python sind unter anderen *for*, *if*, *elif*, *in*, *try*, *pass*, *except*, *from*, *def* und *print*.

3.2.3.1.3.3 Funktionsaufrufe

Durch Funktionen werden Teilaufgaben gelöst. Beim Funktionsaufruf übernimmt die Funktion gewisse Werte und liefert einen Wert als Ausgabe zurück. Argumente oder aktuelle Parameter nennt man die Werte, die einer Funktion übergeben werden.

```
1 #Funktionsaufruf len als Name der Funktion
2 #Zeichenkette „Test“ als Argument übergeben
3 >>> len („Test“)
4 #Funktion liefert Länge der Zeichenkette
5 >>>4
```

3.2.3.1.3.4 Methodenaufrufe

Methoden sind an ein bestimmtes Objekt gebundene Funktionen. Prinzipiell erfolgt ein Methodenaufruf mit dem Namen des Objektes, gefolgt von einen Punkt und der eigentlichen Methode auf das Objekt. Optional können der Methode Argumente in Klammern übergeben werden.

```
1 #Erzeugen eines Objektes namens liste
2 >>> liste = [2, 4, 6, 7, 3]
3 #Methodenaufruf
4 #reverse() liefert liste in umgekehrter Reihenfolge
5 >>> liste.reverse()
6 #Ausgabe des Objektes
7 >>> liste
8 [3, 7, 6, 4, 2]
```

3.2.3.1.3.5 Import-Anweisungen

Standardfunktionen sind fester Bestandteil der Programmiersprache und immer verfügbar. Erweitert werden diese durch speziellere Funktionen in Modulen. Durch die Import-Anweisung können Module eingebunden werden. Verschiedene Varianten stehen dem Programmierer hierbei zur Verfügung.

```
1 >>> from modulname import funktionsname
```

Diese Syntax erlaubt es, gezielt den Namen der benötigten Funktion in den lokalen Namensraum zu importieren.

Um alle Objekte eines Moduls einbinden zu können, wird anstelle des Funktionsnamen's ein * eingesetzt. Als weitere Möglichkeit besteht ein Import des gesamten Modul's.

```
1 #Einbinden aller Objekte aus dem Modul
2 >>> from modulname import *
3 #Import des gesamten Moduls
4 >>> import modulname
```

3.2.3.2 Bereitgestellte Bibliotheken der XBMC Python Engine

Unabhängig vom standardmäßigen Python-Interpreter nutzt XBMC seinen eigenen Built-In-Interpreter. XBMC Python ist eine Implementation von der etwas in die Jahre gekommenen Python Version 2.4. Genutzt werden alle Standardbibliotheken von Python 2.4 sowie hauseigene Bibliotheken. Diese Bibliotheken (Module) sind für die Implementierung benutzerdefinierter Funktionalitäten in XBMC auf Basis von Python erforderlich. Drei Module bieten XBMC-spezifische Funktionalitäten an. Zusammenfassend sind diese Module mit einer kurzen Beschreibung in der nachfolgenden Tabelle dargestellt [41].

Modul	**Beschreibung**
xbmc	Bietet Klassen und Funktionen für die Mediensteuerung über den Medienplayer sowie Informationen über das abgespielte Medium.
xbmcgui	Bietet Klassen und Funktionen für die Manipulation der grafischen Oberfläche in Form von Fenstern, Dialogboxen und anderen Ereignissen.
xbmcplugin	Bietet Klassen und Funktionen, um Informationen über das XBMC-Standard-Menü durch den Entwickler auszugeben.

Tabelle 3-1: Module der XBMC Python Engine

3.2.4 Add-on's – Erweiterungen für 3rd Party Anwendungen

Add-ons erweitern das XBMC um zusätzliche Fähigkeiten und Funktionen. Die Pakete werden üblicherweise von Drittentwicklern programmiert. Anwender können auswählen, welche Pakete installiert werden sollen, um so XBMC zusätzlichen Nutzen und Flexibilität zu verleihen. Für alle Medienrubriken (Bilder, Musik, Videos und Programme) sind aktuell Erweiterungen verfügbar. Stetig kommen neue Erweiterungen durch private Entwickler hinzu.

Aktiviert beziehungsweise installiert können Add-ons unter dem Menüpunkt *Add-ons → System*. Hier wird ein Überblick über derzeit installierte, für Updates bereitstehende oder Online verfügbare Add-ons gegeben (siehe Abbildung 3-10). Eine weitere Möglichkeit besteht in der Installation aus dem ZIP-Dateiformat heraus. Dabei kann das gepackte Add-on zuerst lokal auf die Festplatte gespeichert werden. Anschließend wird aus XBMC heraus die Erweiterung installiert.

41 vgl. http://wiki.xbmc.org/index.php?title=Python_Development, Stand: 05.11.2010

Abbildung 3-10: Menüpunkt Add-ons

Nach der Installation sind die Erweiterungen unter der spezifischen Rubrik zu finden. Unterstützte Add-ons sind Plugins, Skripte, Skins und Event-Clients.

3.2.4.1 Allgemeiner Unterschied zwischen Plugin und Skript [42]

Im Gegensatz zu einem Skript erweitert das Plugin in der Regel eine vorhandene Software um Funktionen. Dabei wird ein solches durch den Nutzer bei Beitritt eines virtuellen Ordners automatisch über die vorhandene Programmschnittstelle aufgerufen. Anbieter definieren zu ihren Softwareprodukten Schnittstellen, über diese gewünschte Erweiterungen programmiert werden können. Skripte arbeiten unabhängig vom Programm und müssen direkt durch den Nutzer aufgerufen werden. Lediglich die in der API vorhandenen Klassen und Funktionen werden vom Skript genutzt.

Speziell bei XBMC erhält man mit Hilfe eines Skriptes Flexibilität und volle Kontrolle über das XBMC-GUI während Plugins schnell und konsequent Informationen für den Nutzer über die standardmäßige Menüstruktur zur Verfügung stellen.

3.2.4.2 Plugin's [43]

Plugins bieten für Drittentwickler die Möglichkeit, ausgewählte Inhalte über das GUI-Menü-Interface von XBMC darzustellen. Meist werden die Inhalte aus Audio-

42 vgl. http://wiki.xbmc.org/index.php?title=Python_Development, Stand: 05.11.2010

43 vgl. http://wiki.xbmc.org/index.php?title=Plugins, Stand: 06.11.2010

beziehungsweise Video-Streams (Internet TV, Radio Stationen, Podcasts) oder von Bilderportalen (Picasa, Flickr) aus dem Internet bezogen.

3.2.4.3 Skripte [44]

Bei Skripten kombiniert XBMC den Python-Interpreter mit dem Windows XML Application Framework. Darunter kann ein XML-basiertes Widget-Toolkit für die Erstellung von grafischen Bedienungsfenstern verstanden werden. So können Entwickler neue Werkzeuge oder Anwendungen wie Kinoführer, elektronische TV-Programmzeitschriften, E-mail oder Instant Messaging-Clients und vieles mehr erstellen.

3.2.4.4 Skin's [45]

Den Anwendern von XBMC wird eine flexible und robuste grafische Benutzeroberfläche geboten. Dies erlaubt eine persönliche Anpassung des Media Center's. Durch einfaches Downloaden von personalisierten Skins oder eine Implementation der Eigenentwicklung verleiht die XBMC-Skinning-Engine ein erhöhtes Seherlebnis.

3.2.4.5 Event-Client's [46]

Von Event-Client's kann XBMC über einen integrierten Event-Server Softwarekommandos empfangen und ausführen. Bekannte Event-Client-Programme unterstützen Eingabegeräte wie Fernbedienungen und Gamepad-Controller. Mit einer individuellen Programmierung lassen sich die Clients auf die Bedürfnisse der Anwender anpassen. Dafür steht die keymap.xml-Datei zur Verfügung. Keymaps definieren eine Zuordnung der Tasten zu Aktionen in XBMC. Zur Zeit sind viele Client-Softwarepakete für PC's, MAC's, Smartphone's und PDA's verfügbar.

Die Aufgabe des Event-Server's liegt in der Anbindung der Eingabegeräte an XBMC. Geschaffen wurde die Funktionalität für die Anbindung vieler Eingabegeräte über verschiedene Hardwareplattformen. Es sollte ein möglichst einfacher Weg zur Kommunikation mit XBMC realisiert werden. Vergleichbar ist der Event-Server mit der etwas älteren Webschnittstelle HTTP-API.

---

44 vgl. http://wiki.xbmc.org/index.php?title=Scripts, Stand: 06.11.2010

45 vgl. http://wiki.xbmc.org/index.php?title=Add-ons, Stand: 06.11.2010

46 vgl. http://wiki.xbmc.org/index.php?title=Add-ons, Stand: 06.11.2010

4 Systemkonzept

Im folgenden Kapitel wird das Systemkonzept für den Aufbau des XBMC Media Center's als interaktive Multimediaplattform im HTPC vorgestellt. Ausgehend vom geplanten Anwendungsfall wird auf die Entwurfsentscheidung und der Netzwerkinfrastruktur mit den eingesetzten Hardware- und Software-Komponenten aus Telekommunikation und Computersystemen eingegangen. Ebenso erfolgt ein detaillierter Überblick über die verwendete Linuxdistribution freeVDR und den installierten Erweiterungen im Media Center.

4.1 Anwendungsfälle (Use Cases)

Bevor Entscheidungen über eingesetzte Hard- und Software getroffen werden, bedarf es im Vorfeld einer ausreichend detaillierten Abgrenzung gemäß der dem Nutzer angebotenen Dienste in XBMC. Daraus abgeleitet können Komponenten zur Realisierung dieser Diente ausgesucht und zu einen funktionierenden Gesamtsystem integriert werden.

Zur Modellierung eines Anwendungsfalls eignet sich UML. Die standardisierte Modellierungssprache enthält Werkzeuge für die Spezifikation, Konstruktion und Dokumentation von Teilen von Software und Systemen. Anwendungsfälle (engl. Use Cases) stellen Elemente für die Modellierung von Anforderungen an ein System zur Verfügung. [47]

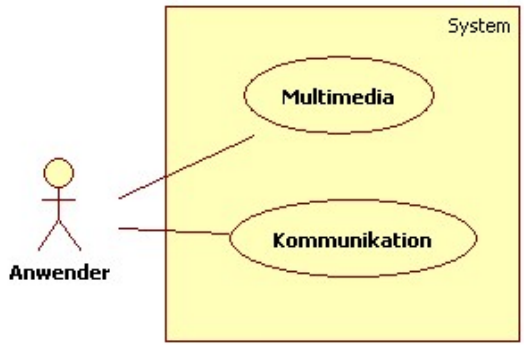

Abbildung 4-1: Angebotene Inhalte in XBMC

Aus Abbildung 4-1 wird der Grundgedanke des Konzeptes ersichtlich. Dem Nutzer sollen im System Inhalte aus zwei Hauptkategorien, Multimedia und Kommunikation, angeboten werden. Das System steht stellvertretend für XBMC. Der Nutzer steht in Assoziation mit

47 vgl. http://de.wikipedia.org/wiki/Unified_Modeling_Language, Stand: 10.11.2010

den im System befindlichen Objekten Multimedia und Kommunikation. Beide Objekte sind Anwendungsfälle im System.

Multimedia wird als Oberbegriff für digitale Medien verwendet. Bezogen auf den aktuellen Anwendungsfall dient es als Hauptkategorie für multimediale Dienste im System. Darunter fallen sowohl standardmäßig installierte Dienste als auch nachträglich hinzugefügte Dienste.

Kommunikation meint den Austausch oder Übertragung von Informationen. Hier repräsentiert das Objekt nachträglich hinzugefügte Telekommunikationsdienste in Form von Add-ons.

Die Dienste hinter den Objekten sind in nachfolgender Abbildung dargestellt. *Live-TV*, *Bilder*, *Musik* und *Videos* sind Anwendungsfälle des Multimedia-Objektes und gleichzeitig als Menüpunkte in XBMC verfügbar. Online Bilder- und Video-Portale bilden einen Untermenüpunkt in Bilder beziehungsweise Videos und werden als Add-ons angeboten. Alle Anwendungen sind im Multimediapaket zusammengefasst.

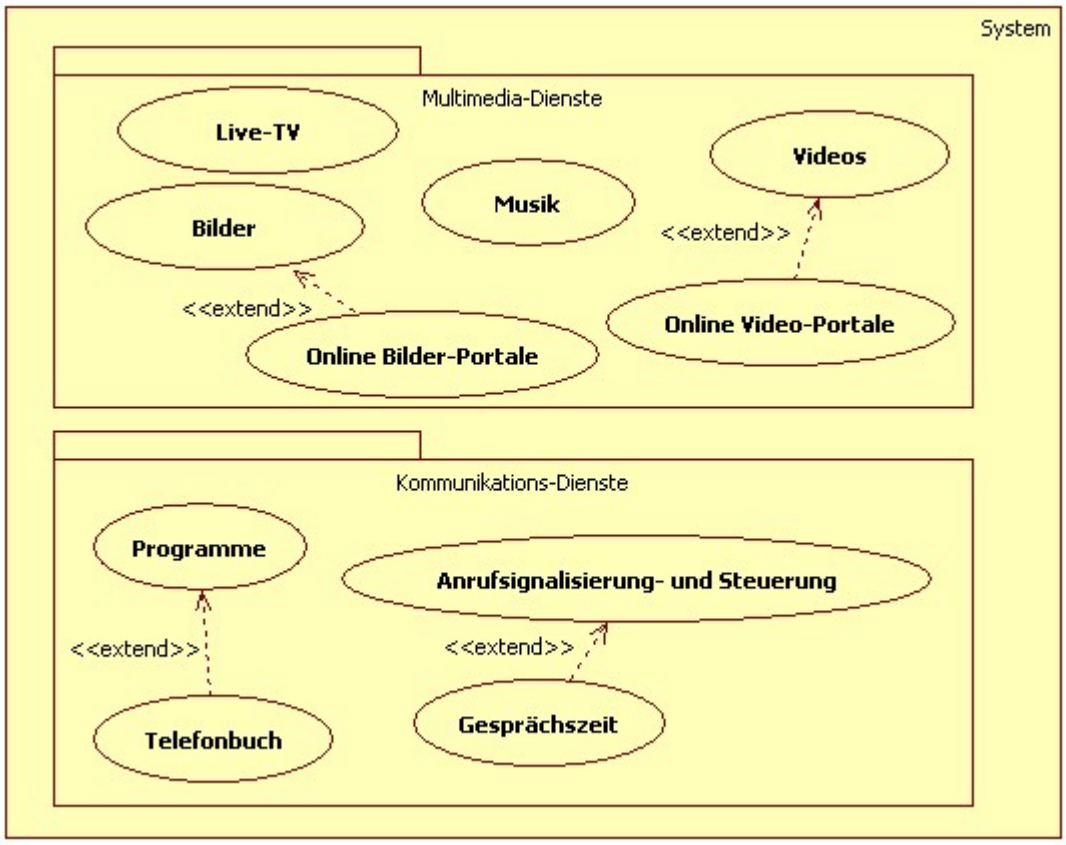

Abbildung 4-2: Dienstpakete im System XBMC

Kommunikationsdienste werden im zweiten Paket zusammenfasst. Über den Programm-Hauptmenüpunkt gelangt der Nutzer zum Telefonbuch und der Gesprächszeit. Die

Anrufsignalisierung und Anrufsteuerung ist nicht direkt über Menüpunkte in XBMC aufrufbar. Vielmehr läuft das Add-on im Hintergrund des Programm's.

Nachstehend werden einzelne Anwendungsfälle aus dem Bereich Kommunikation, die dem Nutzer bezüglich der Anforderungen an das System bereitstehen sollen, aufgezeigt. Anwendungsfälle multimedialer Dienste werden als selbsterklärend angesehen und deshalb nicht weiter erläutert.

4.1.1 Anwendungsfall 1 – Anrufsignalisierung- und Steuerung

Mit dem Erhalt eines Anrufes soll eine Benachrichtigungsbox in XBMC mit Anrufinformationen erscheinen. Da die Rufnummer durch das System übermittelt wird, erfolgt ein Abgleich dieser mit dem Telefonbuch. Befindet sich die Rufnummer im Telefonbuch, wird der dazugehörige Name ausgelesen und im Popup-Fenster zur Anzeige gebracht. Ist die Rufnummer noch nicht im Telefonbuch hinterlegt, erhält der Nutzer die Möglichkeit, die Nummer mit einen Kontaktnamen und Kontaktbild zu verknüpfen. Damit ist der Kontakt im Telefonbuch hinterlegt. Zusätzlich wird das Datum und die Anrufzeit zur Anzeige gebracht. Gerade wiedergegebene Multimedia-Inhalte pausieren bei der Anrufsignalisierung. Ein Gesprächszeitzähler startet automatisch mit aktiver Verbindung zum Teilnehmer. Nach dem Beenden der Verbindung wird die Gesprächszeit über eine Anzeigebox angezeigt.

Steuerelemente in Form von Buttons in der Auswahlbox dienen dem Nutzer für Anrufoptionen. Optionen sind die Annahme und die Ablehnung des Anruf's.

Eingeblendete Benachrichtigungen über den Anrufzustand gehören ebenfalls zum Funktionsumfang.

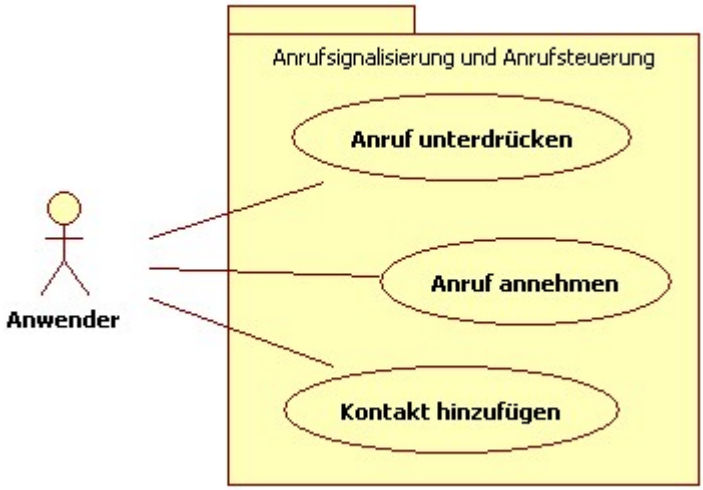

Abbildung 4-3: Use Case Anrufsignalisierung- und Steuerung

4.1.2 Anwendungsfall 2 – Telefonbuch

Eine weitere Anwendung soll das Telefonbuch darstellen. Damit erhält der Nutzer die Möglichkeit, Anrufe aus XBMC heraus zu tätigen. Das Wählen eines im Telefonbuch hinterlegten Kontakt wird mit einem Klick realisiert. Folglich muss der Nutzer nur noch den Hörer des Telefon's abheben oder auf Lautsprecher schalten. Auch hier werden Optionen wie das Löschen oder das Hinzufügen von Kontakten möglich sein. Erreichbar ist das Add-on über den Hauptmenüpunkt *Programme* beziehungsweise *Programm-Plugins*. Zur technischen Umsetzung des Telefonbuch's wird eine Datenbank eingerichtet.

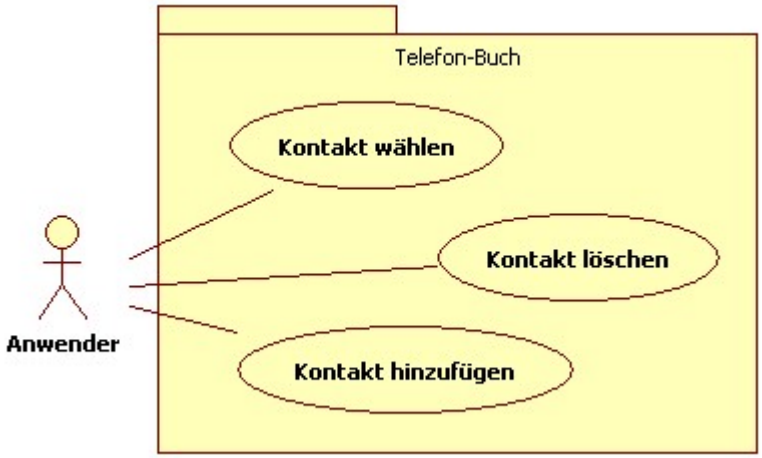

Abbildung 4-4: Use Case Telefonbuch

4.1.3 Anwendungsfall 3 – Gesprächszeit

Dritte und letzte Anwendung soll das Gesprächszeit-Add-on bilden. Das Add-on gibt dem Nutzer die Information auf die bisherige Gesprächszeit aller eingehenden Anrufe und ist gleichermaßen unter den Programm-Plugins zu finden. Ein Löschen und damit das Rücksetzen der Zeit wird ebenfalls vorgesehen. Abgespeichert wird die Zeit ebenfalls in einer Datenbank.

Abbildung 4-5: Use Case Gesprächszeit

4.2 Aktivitäten zwischen Anwender und System

Um die Abläufe zwischen Anwender und System besser darzustellen, sollen UML-Aktivitätsdiagramme eingesetzt werden. Die Aktivitätsdiagramme lassen sich aus den Anwendungsfällen ableiten.

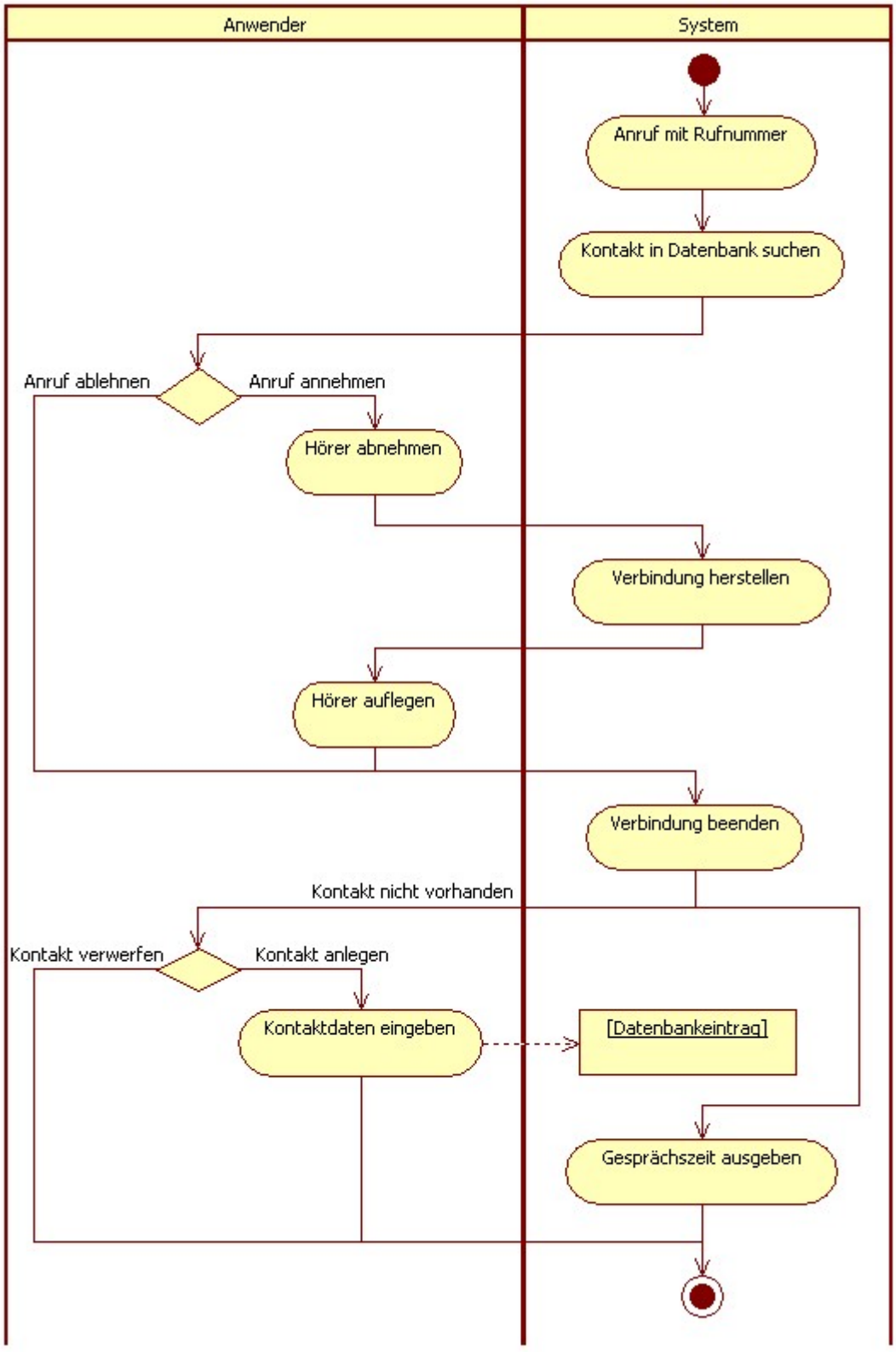

Abbildung 4-6: Aktivitätsdiagramm der Anrufsignalisierung und Anrufsteuerung

Abbildung 4-6 zeigt das Aktivitätsdiagramm der Anrufsignalisierung und Anrufsteuerung. Dabei wird die deutliche Abgrenzung zwischen Anwenderaktivitäten und Systemaktivitäten für die Realisierung der Dienste erkenntlich.

Das nachfolgende Diagramm verdeutlicht die Interaktion zwischen Anwender und System im Telefonbuch-Add-on.

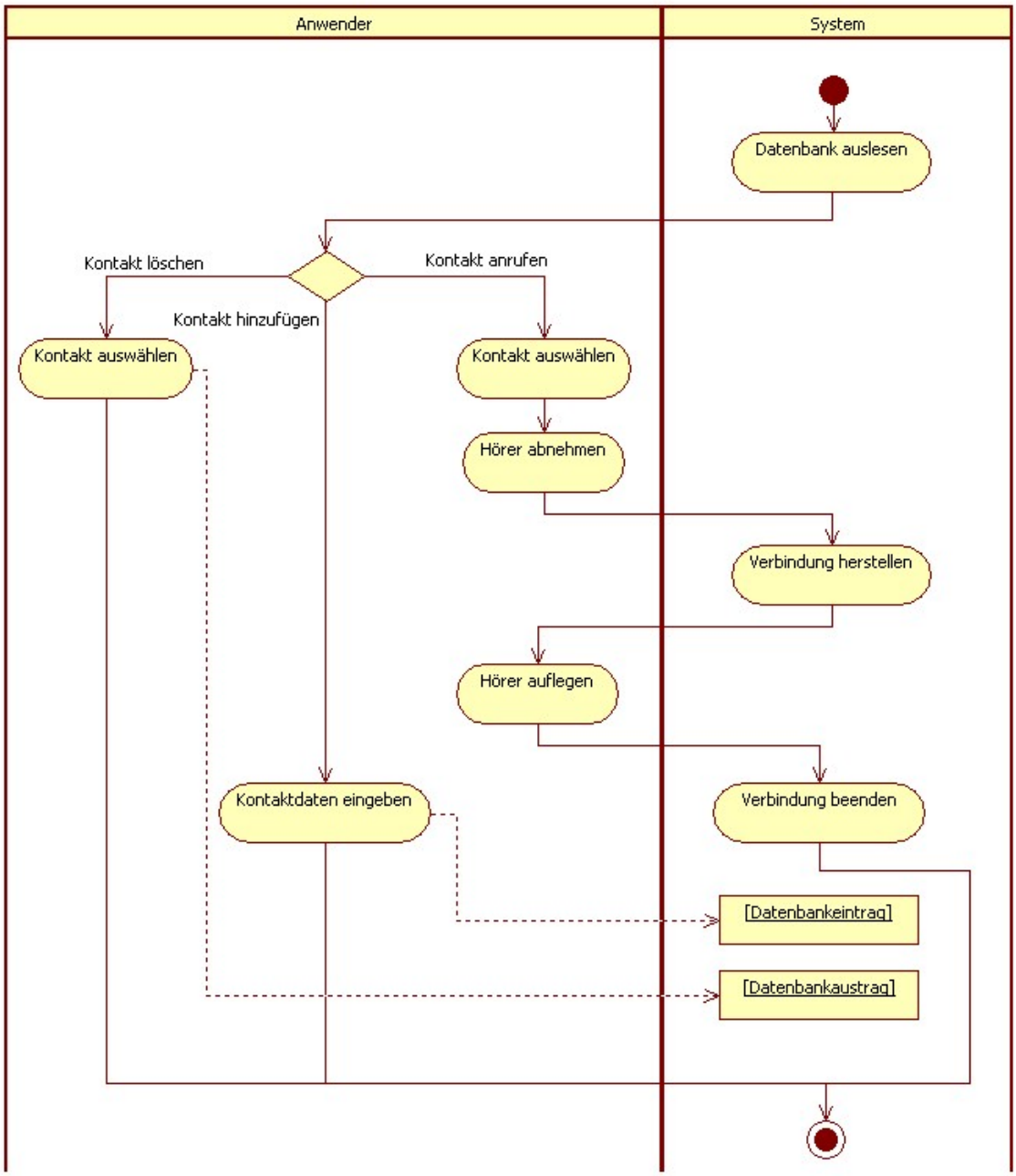

Abbildung 4-7: Aktivitätsdiagramm des Telefonbuch

Letztes Aktivitätsdiagramm beschreibt das Gesprächszeit-Add-on in Abbildung 4-8.

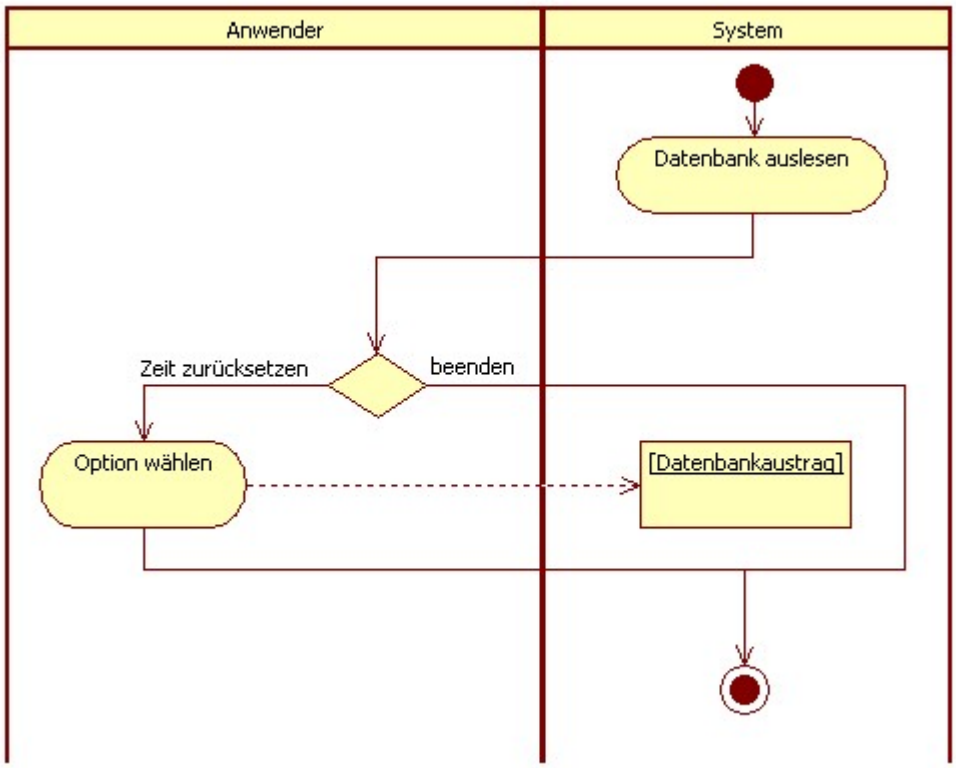

Abbildung 4-8: Aktivitätsdiagramm der Gesprächszeit

4.3 Entwurfsentscheidungen

Entscheidet sich der Anwender gegen ein kommerzielles System, bietet der Open-Source-Markt im Bereich Media Center-Anwendungen eine Reihe von Alternativen an. Gerade durch den offen liegenden Quellcode ist für Entwickler der Anreiz eigene Ideen zu entwerfen, realisieren und einzubringen nahezu unbegrenzt. Eine mehr oder weniger große Community hat sich über die Jahre um die verschiedenen Anwendungen gebildet.

Genau dieser Punkt war im Endeffekt ausschlaggebend für den Entschluss, auf Basis eines HTPC ein interaktives Media Center mit Zugang zu Online-Video- und Bild-Portalen zu entwerfen und mit all seinen benötigten Komponenten aufzubauen.

Als Erstes musste eine Entscheidung bezüglich der Software getroffen werden. Da sich das System als reine Multimediaplattform verhalten soll, war der Entschluss nach Abwägung der aufgeführten Punkte im Kapitelabschnitt 3.1 auf ein Media Center-Betriebssystem als All-in-One-Lösung gefallen. Letztendlich schien nach einer umfangreichen Internetrecherche freeVDR als Linuxdistribution die geeignete Software für weiterführende Entwicklungen und prädestiniert zur Erreichung vorher gesteckter Ziele.

Der nächste Schritt war die Planung der Systeminfrastruktur. In Abhängigkeit der zusätzlich geplanten Eigenentwicklungen mussten Netzwerkinfrastruktur sowie Hard- und Softwarekomponenten geplant beziehungsweise ausgesucht werden. Aufbauend auf dem

Ziel, ein Media Center mit kommunikativen Elementen zu erweitern und lokal angebotene Inhalte mit Webinhalten vermischen zu lassen, musste eine Systeminfrastruktur bestehend aus Telekommunikationskomponenten und EDV-Komponenten entstehen.

4.4 Hardwarekomponenten in der Systeminfrastruktur

Neben dem eigentlichen Media Center bedarf es angesichts der gesteckten Ziele an zusätzlichen Komponenten. Basierend auf einen möglichst verbrauchernahen Anwendungsfall wurde die Telekommunikations- und EDV-Hardware ausgewählt und in das Konzept integriert.

Herzstück des System's bildet das HTPC unter der Linux-Distribution freeVDR. Dem Anwender sollen über das darin enthaltene XBMC sowohl Unterhaltungsdienste als auch Kommunikationsdienste angeboten werden. Bei Unterhaltungsdiensten, die auf lokale Daten zugreifen, bedarf es an keinerlei zusätzlichen externen Komponenten. Lediglich eine für die eigenen Bedürfnisse ausreichend große Festplatte zum Speichern von Videofilmen, Musikdateien oder Bildern ist vonnöten. Mit Hilfe eines DVB-T USB-Stick wird dem HTPC Live-TV zur Verfügung gestellt.

Um über XBMC auf Webdienste zugreifen zu können, braucht es wiederum eine Verbindung in das Internet. Hier steht dem Anwender das Hochschulnetz zur Verfügung. Zusätzlich dient das Netz als Verbindung zwischen dem CTI-Server und dem HTPC (siehe Abbildung 4-9). Das Hochschulnetz der HS Mittweida ist Bestandteil des Deutschen Forschungsnetz (DFN) und verfügt über Knotenpunkte für eine Anbindung in das Internet. Daher verfügt das Hochschulnetz über eine ausreichend hohe Datenübertragungsgeschwindigkeit auch außerhalb des lokalen Netzwerk's.

Zur Bereitstellung von Kommunikationsdiensten dient die Telekommunikationsanlage, kurz TK-Anlage. Die Entscheidung viel auf eine herkömmliche TK-Anlage der Firma elmeg als Gateway in das Fernmeldenetz. Ein wichtiges Auswahlkriterium war die Unterstützung des Telephony Application Programming Interface (TAPI) durch die Anlage. TAPI verbindet die TK-Anlage mit dem IP-Netzwerk und ermöglicht so computerunterstützte Telefonie. Dabei wird auch von Computer Telephony Integration (CTI) gesprochen. Die Voraussetzung dafür liefert der integrierte IP-Router in der TK-Anlage. Über den integrierten WAN-Port kann die Anlage mittels DSL an das Internet angeschlossen werden. Intern verfügt der Router über zwei 50/100Mbit Ethernet-Anschlüsse für das lokale Netzwerk. Optional wird die Anlage über einen dieser beiden Anschlüsse an den Computer angeschlossen. Eine weitere Möglichkeit zum Anschluss besteht über den integrierten USB-Port. Letztere Option wurde für das Konzept nicht berücksichtigt.

An die Anlage ist ein analoges Telefon und ein ISDN-Telefon angeschlossen. Das ISDN-Telefon wird mittels TAPI überwacht. Über das Endgerät können CTI-Funktionalitäten ausgeführt werden. Außerdem werden alle extern ankommenden Anrufe auf das Telefon geleitet. Obendrein ist das analoge Telefon für die interne Kommunikation installiert.

Angeschlossen ist die TK-Anlage über einen ISDN-S0-BUS mit dem Fernmeldenetz. Externe Anrufe können so über die TK-Anlage an den Nutzer und andersherum gelangen. Ein mögliches Endgerät kann hier das Mobilfunktelefon darstellen.

Computerunterstütze Telefonie stellt der im System befindliche CTI-Server bereit. Der Server erlaubt dem Nutzer aus XBMC heraus, Kommunikationsdienste anzuwenden und verknüpft somit die unterschiedlichen Systembereiche. CTI-Funktionalitäten werden über die Software TapiPhone bereitgestellt. Als Betriebssystem kommt auf dem CTI-Server Microsoft Windows XP mit Service Pack 3 zum Einsatz. Visual Studio 2005 dient als Entwicklungsumgebung von TapiPhone.

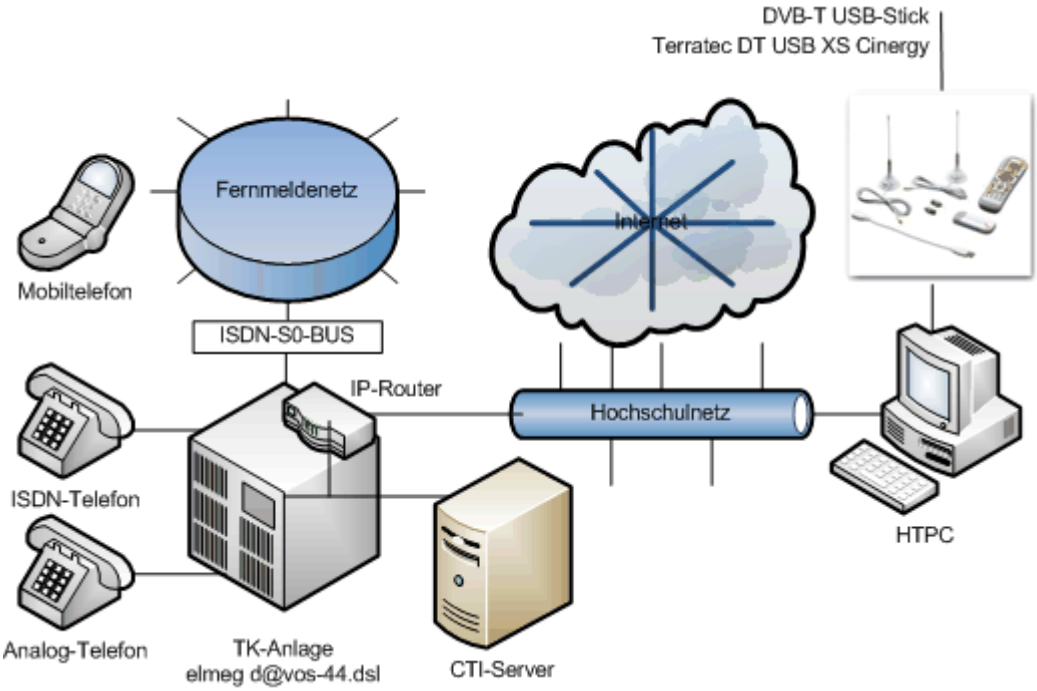

Abbildung 4-9: Hardwarekomponenten und Zugangsnetze in der Systeminfrastruktur

Abbildung 4-9 zeigt alle in der Systeminfrastruktur befindlichen Komponenten.

Auf Basis der weiter unten aufgeführten Grundlagen zu CTI-Architekturen entspricht die konzeptionelle Verbindung zwischen ISDN-Telefon, TK-Anlage und dem CTI-Server als Third Party-Architektur.

4.5 Softwarekomponenten im Systemkonzept

Um die im Systemkonzept eingesetzte Hardware miteinander erfolgreich kommunizieren zu lassen und so ein funktionsfähiges Gesamtsystem zu schaffen, bedarf es an speziell eingerichteter und angepasster Softwarekomponenten (siehe Abbildung 4-10).

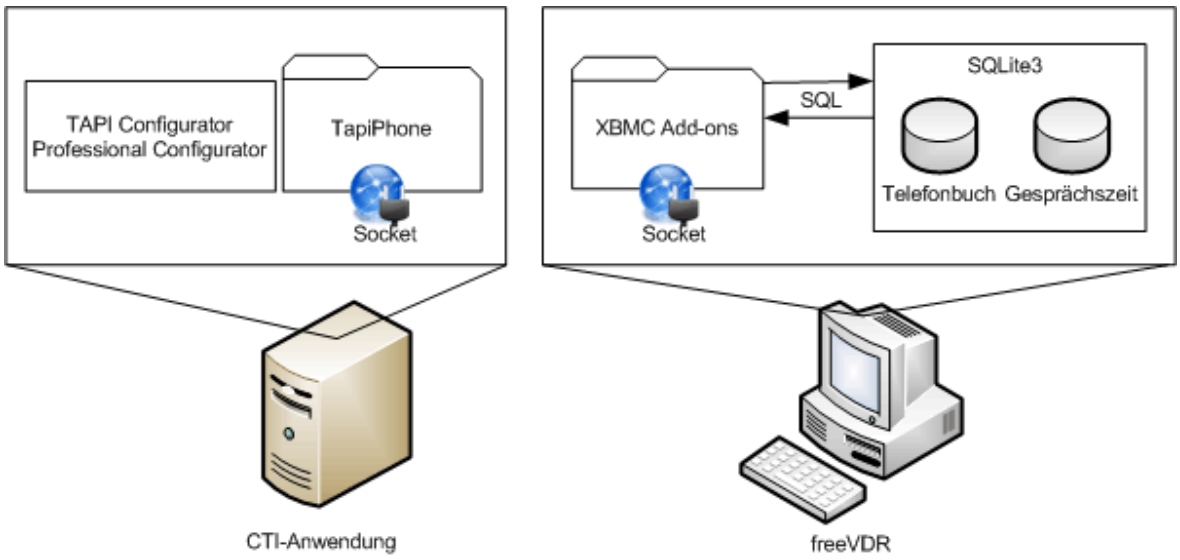

Abbildung 4-10: Softwarekomponenten im Systemkonzept

Im CTI-Server realisiert TapiPhone als Software eine CTI-Anwendung und steuert die Telefonabläufe und gibt Zustandsinformationen des TK-System's an den Anwender aus. Damit stellt die Software das Verbindungsglied zwischen Telekommunikation und Datentechnik dar. Für die Kommunikation über das Hochschulnetz zum Media Center dient ein implementierter Server- und Client-Socket in TapiPhone. Umgesetzt ist die Anwendung in Visual C++. Die beiliegende Software zur Telefonanlage Professional Configurator und TAPI Configurator wird für die Konfiguration der Anlage beziehungsweise des TAPI-Treiber's benötigt.

Als HTPC-Software wird freeVDR verwendet. Die Distribution enthält neben einer minimalistischen Bedienoberfläche Xfce und einen VDR das XBMC Media Center und somit die Voraussetzung zur Umsetzung der Aufgabenstellung. Zur Bereitstellung der Anrufsignalisierung, Anrufsteuerung, des Telefonbuch's und des Gesprächszeitzähler's in XBMC dienen in Python-Linux geschriebene Add-ons in Form von Skripten. Diese werden beim Start von XBMC beziehungsweise bei Bedarf geladen. Server-Socket und Client-Socket dienen als Kommunikationsendpunkte und sind in den dafür vorgesehenen Skripten verankert. Grundlage für die Implementierung des Telefonbuch's in XBMC stellt für die Telefonbucheinträge SQLite3 als Datenbanksoftware dar. Mittels SQL-Befehlen werden Datenbankeinträge ausgelesen, hinzugefügt oder bearbeitet. Das Abspeichern der Gesprächszeit erfolgt ebenfalls über die Datenbanksoftware.

Im Endeffekt soll die Software durch den Prozessor ausgewertet und die softwaregesteuerten Hardwarekomponenten in ihrer Arbeit beeinflussen und steuern. Diesbezüglich sind im Vorfeld grundlegende Sachverhalte zu Programmierschnittstellen, CTI-Architekturen, Socket's und Datenbanksystemen, im speziellen SQLite3, zu klären.

4.5.1 Grundlagen

4.5.1.1 Telephony Application Programming Interface (TAPI)

TAPI bezeichnet eine Programmierschnittstelle für Telefonieanwendungen. Entwickelt wurde die Programmierschnittstelle von Microsoft und Intel und fand seit dem Betriebssystem Microsoft Windows 95 einen festen Platz im System. Über TAPI können Gesprächssteuerungen realisiert werden. Dabei lassen sich Gesprächsverbindungen, Datenverbindungen oder Endgeräte-Hardware (Display, Lautsprecher) steuern. [48]

Hinter TAPI verbirgt sich das Windows Open System Architecture (WOSA)-Prinzip. WOSA erlaubt eine Trennung zwischen Anwendung (Applikation) und Dienstanbieter (SP) über eine zwischengeschaltete Instanz, der TAPI.dll. Hauptaufgabe der TAPI.dll liegt in der Vermittlung und Koordinierung von Applikation zu SP. Möchte z.B. eine Applikation ein Fax verschicken, übernimmt TAPI.dll die Vermittlung zum geeigneten SP. Die TAPI.dll stellt mittels TAPI eine Vielzahl von Funktionen den Applikationen bereit. Wiederum wird die Schnittstelle zwischen der TAPI.dll und den SP's Telephony Service Provider Interface (TSPI) bezeichnet. Abbildung 4-11 soll diesen Zusammenhang bildlich verdeutlichen.

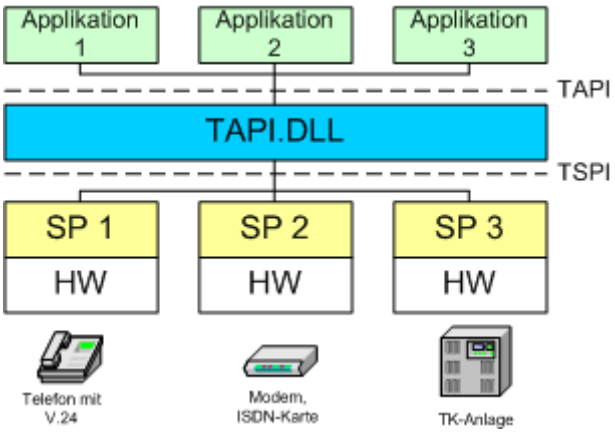

Abbildung 4-11: TAPI.dll als Vermittlung zwischen Applikation und Service Provider

(Bildquelle: http://telecom.htwm.de/telecom/praktikum/CTI_TAPI/images/TAPI-Konzept_screen.gif, Stand: 25.11.2010)

Unterteilen lässt sich der Funktionsumfang in vier Bereiche. Mit den Basic Telephony Services werden grundlegende Funktionen wie Verbindungsaufbau und Verbindungsabbau abgedeckt und sind von allen SP's anzubieten. Hingegen wird der Funktionsumfang bei den sogenannten Supplementary Telephony Services optional angeboten. Darunter fallen z.B. Rufumleitung und Halten des Gespräch's. Weitere Bereiche sind Extended Telephony Services und Assisted Telephony Services. [49]

---

48 vgl. http://de.wikipedia.org/wiki/Telephony_Application_Programming_Interface, Stand: 25.11.2010

4.5.1.2 Computer Telephony Integration (CTI) [50]

Damit wird die Verbindung von Telekommunikationskomponenten mit dem Computersystem und dessen Infrastruktur bezeichnet. Als Telekommunikationskomponenten werden unter anderem TK-Anlagen, Telefone und das Telekommunikationsnetzwerk verstanden. CTI unterstützt eine Vielzahl von Telefonvorgängen und bringt somit herkömmliche Telefonfunktionen auf eine spezielle Computeranwendung. So können normale Gesprächsabläufe von der Wahl des Teilnehmer's über die Beendigung des Gespräch's von einer externen Anwendung ausgeführt werden. Aber auch umfangreiche Zusatzfunktionen wie Informationen zum Anrufer, der Weiterleitung des Gesprächs oder die Gesprächsdauer können mit CTI erfasst und ausgewertet werden.

Darüber hinaus unterscheidet CTI noch unterschiedliche Architekturen. Bei der sogenannten First Party-Architektur besteht eine direkte Verbindung zwischen Computer und Telefon. In dem Fall steuert die Applikation das Telefon. In Kopplung mit einer TK-Anlage wird bei der First Party-Architektur noch zwischen phone centric und computer centric unterschieden. Bei der direkten Verbindung zwischen dem am Computer angeschlossenen Telefon und der TK-Anlage wird von phone centric gesprochen. Der Applikation stehen demzufolge nur auf das Telefon begrenzte Funktionen bereit. Hingegen dient der Computer mit seiner Applikation bei computer centric als Verbindung zwischen Telefon und TK-Anlage. Er hat die Möglichkeit das Telefon zu steuern und bestimmte Funktionen von der Applikation ausführen zu lassen.

Neben der First Party-Architektur definiert CTI die Third Party-Architektur. Wichtigstes Merkmal ist hier die physikalische Trennung zwischen Telefon und Computer. Ein CTI-Server stellt die Verbindung zwischen der TK-Anlage und den im Netzwerk befindlichen Computer bereit. Mittels des Servers lassen sich TK-Anlage und angeschlossene Telefone überwachen und steuern.

Die Verbindung zwischen der TK-Anlage und dem Computersystem bildet eine Schnittstelle, der CTI-Link. An dieser Schnittstelle übernehmen Standards wie TAPI die Signalisierung eingehender oder ausgehender Anrufe zwischen der CTI-Anwendung und dem Computerprogramm.

4.5.1.3 CTI-Anwendung über TAPI [51]

Demzufolge stellt TAPI der CTI-Anwendung ihre Funktionen bereit. Möchte nun eine CTI-Anwendung auf die TAPI-Funktionen zugreifen, muss zu Beginn eine Verbindung zwischen der CTI-Anwendung und dem TAPI Service Provider initialisiert werden.

---

49 vgl. http://telecom.htwm.de/telecom/praktikum/CTI_TAPI/CTI_TAPI.htm, Stand: 25.11.2010

50 vgl. http://telecom.htwm.de/telecom/praktikum/CTI_TAPI/CTI_TAPI.htm, Stand: 25.11.2010

51 vgl. http://telecom.htwm.de/telecom/praktikum/CTI_TAPI/CTI_TAPI.htm, Stand: 25.11.2010

Initialisiert wird durch die TAPI-Funktion *lineInitialize*. Um eine erfolgreiche Initialisierung durchzuführen, müssen Übergabeparameter der Funktion übergeben werden. Ein besonders wichtiger Parameter ist hier *LINECALLBACK fnCallback*. Damit wird die Adresse auf die Callback-Funktion, welche alle Nachrichten vom Treiber empfängt, übergeben. Nach erfolgreicher Initialisierung liefert die Funktion *lineInitialize* den Wert Null zurück, andernfalls einen negativen Wert.

Nun muss nach der Initialisierung der TAPI-Session diese geöffnet werden. Dem bestimmten Dienst (Daten, Sprache usw.) erfolgt eine Übergabe des entsprechenden Line-Device. Zum Öffnen des Line-Device wird die Funktion *lineopen* verwendet. Auch hier sind Übergabeparameter erforderlich. Wie bei der Funktion *lineInitialize* liefert die Funktion bei erfolgreichen Öffnen den Wert Null zurück, ansonsten einen negativen Wert.

Nachdem das LINE-Objekt initialisiert und geöffnet wurde, können nun zum Ausführen und Beenden eines Call's (Anruf) die dazugehörigen TAPI-Funktionen aufgerufen werden. Hierfür stehen die Funktionen *lineMakeCall* und *lineDrop* für Ruf erzeugen beziehungsweise Ruf beenden zur Verfügung. Abschließend ist noch zu sagen, dass die Funktion *lineMakeCall* einen Parameter zurückliefert, welcher für anschließende Steuerungen den Anruf eindeutig beschreibt. Eine anschließende Steuerung kann Ruf beenden sein. Liefern die Funktionen in beiden Fällen einen negativen Wert, ist ein Fehler aufgetreten.

Die nachfolgende Abbildung veranschaulicht einen häufigen Session-Ablauf :

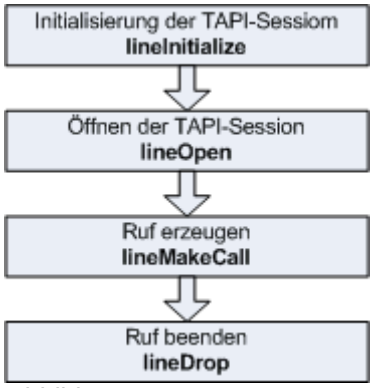

Abbildung 4-12: TAPI-Funktionen

4.5.1.4 Socket's [52]

Socket's sind Kommunikationsendpunkte. Mittels derer können verteilte Anwendungen Transportdienste auf den Schichten Eins bis Drei und Vier nutzen. Unterschieden werden bei Socket's zwischen dem Server-Socket und dem Client-Socket. Der Server-Teil der

52 vgl. https://www.telecom.hs-mittweida.de/index.php?eID=tx_nawsecuredl&u=0&file=fileadmin/verzeichnisfreigaben/telecom/winkler/vorlesungen/sockets.pdf&t=1295889766&hash=3153bf6f6e19c953b006e568c91818cc, Stand: 29.11.2010

Anwendung erstellt einen Server-Socket. Dieser wartet auf Assoziationen. Der Client-Teil errichtet einen Client-Socket, der aktiv Assoziationen zum Server-Socket herstellt (siehe Abbildung 4-13).

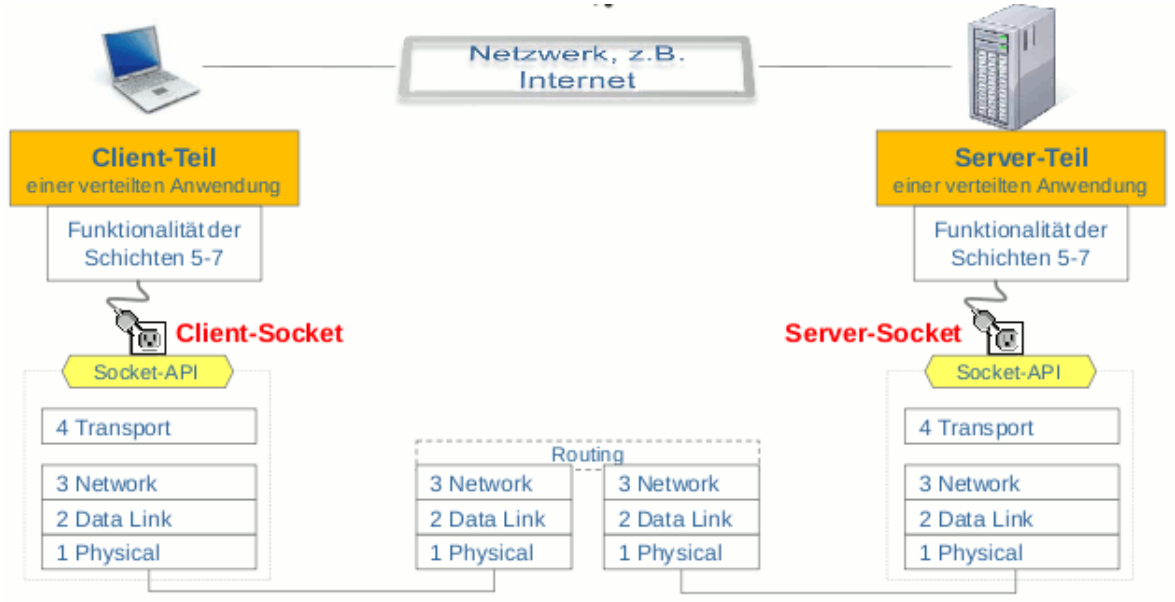

Abbildung 4-13: Client-Socket und Server-Socket

Socket's sind für viele Betriebssysteme verfügbar. Funktionen werden über eine API zur Verfügung gestellt. In UNIX beziehungsweise LINUX sind sie Teil des Betriebssystems. Anders in Windows, dort werden sie als Bibliothek, genauer gesagt als Dynamic Link Library (DLL) [siehe Glossar] im System abgelegt.

Basierend auf dem Unix-Eingabe-Ausgabe-Paradigma open → read/write → close erfolgt die Socket-Nutzung.

Socket-APIs können mehrere Netzwerkprotokolle unterstützen. AF_INET steht für Address Family InterNet, Version 4. Dahinter verbirgt sich das Netzwerkprotokoll IPv4. Darüber hinaus gibt es noch AF_INET6 (Address Family InterNet, Version 6), AF_UNIX (Address Family UNIX), AF_IPX und viele mehr.

Ausgehend von AF_INET werden drei Verbindungstypen unterstützt:

- SOCK-STREAM

  Bei SOCK_STREAM besteht eine verbindungsorientierte Assoziation zwischen den Anwendungen. Der Kommunikationsablauf erfolgt auf dem Weg: Socket errichten → Verbindungsaufbau → Kommunikation → Verbindungsabbau. Die Übertragung der Bytes erfolgt über das TCP-Protokoll sequenzweise. Jede Sequenz wird nummeriert und vom gegenüberliegenden Partner quittiert. Kommt es zu keiner Quittierung der Sequenz erfolgt das nochmalige Senden dieser. Auf der Empfangsseite wird anhand der Nummerierung die ursprüngliche Reihenfolge

wiederhergestellt. Nachteil dieses Verbindungstyps ist der langsame Datendurchsatz. Jedoch kann es zu keinem Paketverlust kommen.

- SOCK_DGRAM

  Eine verbindungslose Assoziation kennzeichnet SOCK_DGRAM. Hier ist nur eine Errichtung des Socket's mit folgender Kommunikation notwendig. Durch die datagram-weise Byteübertragung über das UDP-Protokoll entfällt die Nummerierung der Sequenzen und somit eine Datensicherung. Anwendung findet der Typ bei Echtzeitdaten, wo es auf schnellen Datendurchsatz ankommt.

- SOCK_RAW

  Unter SOCK_RAW erfolgt eine direkte Nutzung der verbindungslosen IP-Datagram-Übertragung.

Damit mehrere Anwendungen gleichzeitig die TCP-Instanz beziehungsweise die UDP-Instanz verwenden können, hat man Portnummern eingeführt. Ports sind Teil einer Adresse und ordnen einem Netzwerkprotokoll seine Datensegmente zu. So können Protokolle auf höheren OSI-Schichten adressiert werden. Wenn man TCP und UDP zusammenfasst, gibt es insgesamt 2^{16} Portnummern, also von Null bis 65535. Verwaltet werden die Bereiche durch die IANA. IANA untergliedert insgesamt drei Portbereiche. Die Ports von Null bis 1023 sind für allgemein anerkannte Dienste reserviert. Diesem Portbereich wurde die Bezeichnung Well Known Ports gegeben. Sogenannte Registered Ports sind für proprietäre Dienste von den Portnummern 1024 bis 49151 reserviert. Schließlich folgen die Ports für den privaten Gebrauch von 49152 bis 65535.

WS-API oder WSA heißt das Windows-API für Socket-Unterstützung. Die Windows-Socket-API unterstützt alle Berkeley Software Distribution (BSD)-Socket-Funktionen und weitere Funktionen. Möchten nun zwei Anwendungen miteinander über Socket's kommunizieren, sind verschiedene Socket-Dienste und deren Funktionen im Vorfeld abzurufen. Die nachfolgende Darstellung zeigt den Ablauf von der Errichtung eines Socket's bis hin zum Abbau des Socket's.

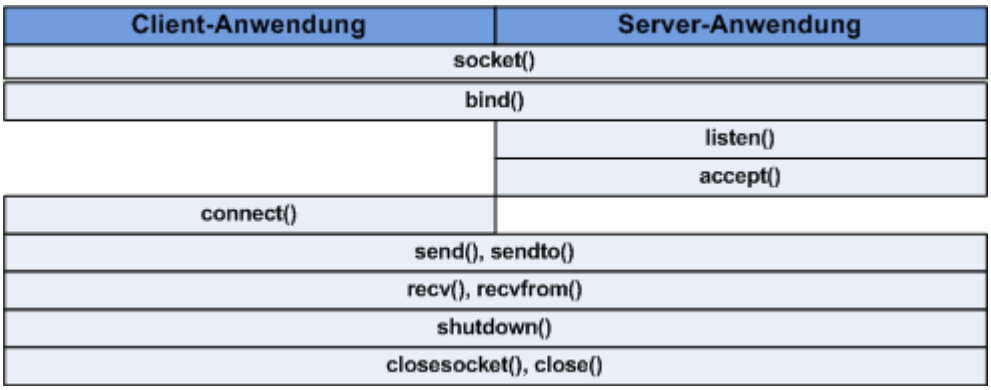

Abbildung 4-14: wichtige Socket-Funktionen

Zuerst muss der Socket initialisiert werden. Mit dem Funktionsaufruf *socket()* errichtet sowohl der Client als auch der Server einen Socket. Zusätzlich muss der Parameter für das Adressenformat (AF_INET, AF_INET6...) und dem Socket-Typ (SOCK_STREAM, SOCK_DGRAM...) der Funktion übergeben werden.

Als nächstes muss der angelegte Socket mit *bind()* an eine vom Nutzer gewünschte lokale IP-Adresse und lokalen Port gebunden werden. IP-Adresse und Port sind als Parameter wiederum der Funktion zu übergeben. Bei Server-Anwendungen ist dieser Schritt notwendig. Hingegen ist bei einer Client-Anwendung der Schritt optional.

Damit die Server-Anwendung Daten von der Client-Anwendung empfangen kann, sind serverseitig Verbindungsbereitschaft herzustellen und Verbindungsassoziationen zu akzeptieren. Mit *listen()* stellt die Server-Anwendung Assoziationsbereitschaft her. Dabei wird der Socket in ein Server-Socket konvertiert. Darauf hin wartet die Server-Anwendung auf Verbindungen. Durch *accept()* erfolgt die Annahme von Clientverbindungen. Bei Annahme wird ein neuer Socket zum anrufenden Client errichtet und der Server-Socket ist folglich zur Entgegennahme weiterer Verbindungen bereit.

Clientseitig findet nun die Verbindungsherstellung mit der Server-Anwendung statt. Über *connect()* erfolgt eine Assoziation zwischen dem lokalen Host und Port und dem entfernten Host und Port. Bei einer TCP-Verbindung wird die Assoziation in Form einer Verbindung errichtet. Andererseits werden bei UDP lediglich die lokale als auch die entfernte IP-Adresse und Port festgelegt. Beide Adressen werden der Funktion als Parameter übergeben.

Für die Datenübertragung sind serverseitig und clientseitig folgende Funktionen bereitgestellt. Über *send()* beziehungsweise *sendto()* werden Daten in einen vollständig adressierten Socket gesendet. Letztere Funktion erlaubt das Senden unter Angabe der entfernten IP-Adresse. Zum Empfang der Daten aus einen vollständig adressierten Socket werden *recv()* und *recvfrom()* verwendet. Bei *recvfrom()* wird nach dem Funktionsaufruf und anschließender Abarbeitung die Absenderadresse in der Struktur abgelegt.

Nach der Datenübertragung kann die Verbindung beendet und der Socket geschlossen werden. Mit *shutdown()* beendet die Server- oder Client-Anwendung die Verbindung halb-seitig. Allerdings ist dies nur bei einer TCP-Verbindung möglich. Zu guter Letzt bewirkt *closesocket()* beziehungsweise *close()* eine vollständige Beendigung und Schließung des Socket's.

In Abhängigkeit verschiedener Programmiersprachen können die Funktionsnamen und Methodennamen abweichen. Dafür sind die dazugehörigen Bibliotheken und Module verantwortlich. Bei der späteren Implementierung des Systemkonzeptes wird dies deutlich. Das Konzept bleibt jedoch immer das selbe.

Die Nutzung des Socket's über diese Funktionen hat einen entscheidenden Nachteil. Funktionen wie *accept()*, *send()* und *recv()* blockieren die Programmausführung, solange

sie nicht ordnungsgemäß abgelaufen sind. Wenn der Server nur sehr langsam auf Anfragen reagiert, können Verbindungs- und Aufbauphasen mitunter zeitlich sehr lang werden. Während dieser Zeit kann das Programm keine weiteren Prozesse ausführen. Im ungünstigsten Fall kann solch eine Blocking-Funktion sogar zum vollständigen Einfrieren des Programm's beitragen. Um dies zu vermeiden gibt es verschiedene Lösungsansätze, einen nicht-blockierenden Socket zu errichten.

4.5.1.4.1 Socket-API unter Python Linux [53]

Grundlegende Netzwerkfunktionalität bietet unter Python das Modul *socket*. Wie in vielen anderen Programmiersprachen auch bildet das Modul die standardisierte Socket-API ab. Socket-API meldet das Programm, welches über die Netzwerkschnittstelle Daten Empfangen oder Senden möchte, am Betriebssystem an. Anschließend stellt das Betriebssystem ein Socket dem Programm zur Verfügung.

Eingebunden wird das Modul über die Importanweisung und dem Namen des Modul's.

```
1 #Einbinden des Moduls
2 import socket
```

In Folge dessen stehen dem Programmierer innerhalb der Socket-Klasse Funktionen und Methoden zur Initialisierung, zum Verbindungsaufbau, für Statusabfragen, zur Datenübertragung und zum Verbindungsabbau zur Verfügung. Anhand eines Server-Socket sollen diese Schritte verdeutlicht werden:

```
3 #Socket initialisieren
4 s = socket.socket(Adressenformat, Socket-Typ)
5 #Socket an lokale IP-Adresse und Port binden
6 s.bind((Host,Port))
7 #Assoziationsbereitschaft herstellen
8 s.listen()
9 #Annahme von Clientverbindungen
10 connect = s.accept()
11 #Daten empfangen
12 data = connect.recv(1024)
13 #Socket schließen
14 s.close()
```

Erkenntlich ist das grundlegende Konzept der Socket-Implementierung und Nutzung.

53 vgl. http://openbook.galileocomputing.de/python/python_kapitel_20_001.htm, Stand: 02.12.2010

4.5.1.4.2 Nichtblockierende Socket's [54]

Eine Möglichkeit, den Socket in einen nicht-blockierenden Zustand zu versetzen, bietet Polling. Durch Polling werden in zyklischen Zeitintervallen Aufrufe zum aktuellen Status durchgeführt. Darin können z.B. Informationen zur Lesbarkeit oder Beschreibbarkeit des Socket's abgerufen werden.

Ähnlich dem Polling verhält es sich mit bestimmten Socket-Flags. Flags können zwei Werte, abhängig der Programmiersprache, annehmen. Üblicherweise nehmen die Flags *true* oder *false* an. Wird das Flag aufgrund von unvorhergesehenen Problemen auf *false* vor einem Methodenaufruf gesetzt, so blockieren diese bei ihrem Aufruf das Programm nicht mehr. Später können durch Ausnahmen bestimmte Fehlerbehandlungen durchgeführt werden.

Effektiver ist das Arbeiten mit Threads. Diese ermöglichen durch Multitasking mehrere Abarbeitungen im Programm gleichzeitig durchzuführen. So können vom Server-Socket alle Clientanfragen in separaten Threads verwaltet werden. Asynchrone Socket's verwenden zur Verarbeitung von Netzwerkverbindungen solche Threads.

Asynchrone Socket's stellen auch den Lösungsansatz im Systemkonzept dar. Deshalb erfolgt in den folgenden Kapitelabschnitten eine einleitende Erklärung zu den Klassen und Modulen unter Visual C++ und Python.

4.5.1.4.3 CAsyncSocket – asynchrone Socket-Implementierung unter Visual C++ [55]

Visual C++ definiert zur asynchronen Socket-Implementierung eine eigene Klasse *CAsyncSocket*. Die Klasse kapselt Windows Socket API-Funktionen und bietet eine objektorientierte Abstraktion für Programmierer, die Windows Socket's in Verbindung mit MFC wollen. Erhöhte Verwaltungsaufgaben gegenüber der normalen *CSocket*-Klasse sind hier zu beachten. So müssen Behandlungen zu Blocking, Byte-Reihenfolge und zur Umwandlung von Unicode und Multibyte-Zeichensätzen aufgeführt werden.

Um ein *CAsyncSocket*-Objekt nutzen zu können, muss ihr Konstruktor aufgerufen werden. Danach gilt sowohl für die Client-Anwendung als auch die Server-Anwendung der Funktionsaufruf *Create()* zum Errichten eines Socket's. Anschließend erfolgt auf Seiten des Server's die *Listen()*-Funktion und auf Clientseite der Aufruf der *Connect()*-Funktion. Für die Annahme einer Clientverbindung sollte die Server-Anwendung den Funktionsaufruf *Accept()* durchführen. Letztendlich wird mit Aufruf der *Close()*-Funktion der Socket geschlossen.

---

54 vgl. http://www.codeplanet.eu/tutorials/csharp/4-tcp-ip-socket-programmierung-in-csharp.html?start=4, Stand: 02.12.2010

55 vgl. http://msdn.microsoft.com/en-us/library/3d46645f%28v=vs.80%29.aspx, Stand: 05.01.2011

Auch hier ist wieder das allgemeine Konzept der Windows Socket-API zu erkennen. Lediglich weichen die Funktionsnamen etwas ab.

4.5.1.4.4 asyncore – asynchrone Socket-Implementierung unter Python [56]

Asynchrone Socket-Dienste für Client- und Server-Anwendungen werden mit diesen Modul in Python angeboten und bilden eine Alternative zu Multi-Threading-Programme. Viele Probleme lassen sich mit dem asyncore-Modul in Hinblick auf High-Performance-Netzwerke lösen. Dem Modul wird die Unterstützung des select() System-Aufruf in der Eingabe/Ausgabe (I/O)-Bibliothek des Betriebssystem's vorausgesetzt. Mit der ist es möglich, mehrere Kommunikationskanäle auf einmal zu verwalten; eine andere Arbeit verrichten während die Eingabe/Ausgabe in den Hintergrund gerückt wird.

Der Grundgedanke hinter dem Modul liegt in der Erzeugung einer oder mehrerer Netzwerkkanäle, Instanzen der Klasse asyncore.dispatcher. Nach Erzeugung werden die Kanäle zu einer allgemeinen Abbildung hinzugefügt und durch die *loop()*-Funktion verwendet. Sobald der erste Kanal beziehungsweise die ersten Kanäle erstellt wurden, ruft die Schleifenfunktion *loop()* die aktivierten Kanaldienste auf. Der Vorgang wird bis zum letzten geschlossenen Kanal fortgeführt (einschließlich der Kanäle, die während asynchroner Dienste zur Abbildung hinzugefügt wurden).

Mit dem Funktionsaufruf *asyncore.loop([timeout[, use_poll[, map[, count]]])* wird eine Abfrage-Schleife betreten. Der *count*-Parameter übergibt der Funktion die Anzahl der Durchgänge bis die Schleife beendet oder alle geöffneten Kanäle geschlossen werden. Standardmäßig ist der Wert auf *None* gesetzt. Dabei endet erst nach der Schließung aller Kanäle die Schleife. Über das *timeout*-Argument werden Timeout-Parameter für den entsprechenden *select()*- oder *poll()*-Aufruf in Sekunden festgelegt. Der Standardwert liegt bei 30 Sekunden. Zu guter Letzt bestimmt der *use_poll*-Parameter die Priorität der beiden *select()* und *poll()*-Funktionen. Wenn der Wert auf *True* gesetzt ist, erhält *poll()* den Vorzug vor *select()*. Jedoch ist der Wert standardmäßig auf *False* gesetzt.

Die dispatcher-Klasse ist eine Verpackung um ein Low-Level Socket-Objekt. Es beinhaltet einige Methoden zur Ereignisbehandlung welche von der *loop()*-Funktion aufgerufen werden. Sogenannte übergeordnete Ereignisse lösen Low-Level Ereignisse zu bestimmten Zeiten oder bei einen bestimmten Verbindungszustand aus. Signalisiert wird alles von der *loop()*-Funktion. Übergeordnete Ereignisse sind *handle_connect()*, *handle_close()* und *handle_accept()*. Impliziert wird *handle_connect()* bei dem ersten Lese- oder Schreibereignis, *handle_close()* bei einem Leseereignis wo keine Daten verfügbar sind und *handle_accept()* bei einem Leseereignis auf einen lauschenden Socket.

Während der asynchronen Verarbeitung werden auf jeden zugeordneten Kanal *readable()*- und *writeable()*-Methoden angewendet, um festzustellen, ob der Kanal-Socket

---

56 vgl. http://docs.python.org/library/asyncore.html, Stand: 05.01.2011

in die Liste von *select()* oder *poll()* für Lese- und Schreibereignisse aufgenommen werden soll.

Wichtige Methoden der Ereignisbehandlung sind *handle_connect()*, *handle_accept()*, *handle_read()*, *handle_write()* und *handle_close()*.

Die Methode *handle_connect()* wird beim Verbindungsaufbau durch den aktiven Socket -Eröffner aufgerufen. Zum Beispiel kann in der Methode eine Protokollaushandlung mit dem entfernten Rechner vollzogen werden. Aufgerufen wird die Methode *handle_accept()* am hörenden Socket (passiven Eröffner), wenn eine Verbindung mit dem entfernten Endpunkt nach dessen *connect()-Aufruf* errichtet werden kann. Die Methode *handle_read()* wird aufgerufen, wenn die *loop()*-Funktion erkennt, dass ein *read()*-Aufruf auf dem Kanal-Socket erfolgen wird. In h*andle_write* können notwendige Methoden zur Pufferung implementiert werden. Nachdem die *loop()*-Funktion einen beschreibbaren Socket zum Beschreiben erkennt, wird diese Methode aufgerufen. Abschließend wird bei Schließung des Socket's die Methode *handle_close()* durch die l*oop()*-Funktion aufgerufen.

Abgesehen von den Ereignismethoden können auch normale Socket-Methoden innerhalb der Klasse asyncore.dispatcher und den Ereignismethoden aufgerufen werden. Dafür stehen z.B. s*end()*, *listen()*, *recv()* oder *bind()* dem Programmierer zur Verfügung.

4.5.1.5 SQLite3 unter Python [57]

Eine Datenbank (DB) stellt die Abstraktionsschicht zwischen dem benutzten Programm und dem physikalischen Massenspeicher dar. DB erleichtern die Verwaltung und Speicherung von großen Datenmengen. Die Kommunikation zwischen dem Programm und der DB geschieht über eine standardisierte Schnittstelle. Neben der DB bildet das Datenbankmanagementsystem (DBMS) zusammen das Datenbanksystem (DBS). Das DBS nimmt Abfragen entgegen und liefert die angeforderten Datensätze zurück. Im Bezug auf DB werden die Abfragen auch Queries genannt. DB, die ihre Bestände in Tabellen organisieren, werden relationale DB genannt. Dafür vorgesehene Abfragen werden in der eigens entwickelten Structured Query Language (SQL)-Sprache realisiert. SQL ist standardisiert und wird von nahezu allen Datenbanksystemen unterstützt. Allerdings implementieren die Systeme immer nur Teilmengen der Sprache und ändern diese geringfügig ab.

SQLite3 ist die Standarddatenbank unter Python und bildet ein sehr einfaches aber dennoch leistungsstarkes DBS. Seine Daten speichert das System in normalen Daten ab. Beim Einbinden des Modul's *sqlite3* stehen dem Programmierer Funktionen und Methoden für das Arbeiten mit einer SQLite3-DB bereit.

---

57 vgl.
http://openbook.galileocomputing.de/python/python_kapitel_19_003.htm#mjea5883d439a425e2f974548b406b56c0, Stand: 03.12.2010

```
1 #Einbinden des Moduls
2 import sqlite3
```

Zweiter Schritt ist der Verbindungsaufbau mit der DB über die *connect()*-Funktion. Als Parameter wird der Pfad zur DB als String oder Unicode-Zeichensatz übergeben.

```
1 #Verbindung zur Datenbank herstellen
2 conn = sqlite3.connect(„Pfad zur Datenbank“)
```

Mit Hilfe eines sogenannten Cursor können die Datensätze verändert oder abgefragt werden. Cursor markieren die aktuelle Bearbeitungsposition innerhalb der DB. Ein neuer Cursor kann mit der *cursor()*-Methode angelegt werden.

```
1 #Cursor-Objekt erzeugen
2 cursor = conn.cursor()
```

Fortan stehen dem Programmierer SQL-Befehle zur Erzeugung neuer Tabellen, zum Schreiben neuer Datensätze in Tabellen und viele weitere Bearbeitungsoptionen bereit. Auf diese Befehle soll hier nicht weiter eingegangen werden. Spezifische Erläuterungen für die Funktionalität der Add-ons findet man unter dem Kapitel 5.

4.5.2 Software des HTPC – freeVDR 3.0

Um möglichst komfortabel und einfach einen VDR-PC zu erhalten, eignet sich das aus einem Hobbyprojekt entstandene freeVDR auf Distributionsbasis. Gleich nach der Installation sollen dem Nutzer grundlegende Funktionen für den Betrieb eines VDR zur Verfügung stehen. Dabei stehen dem Nutzer die aktuelle Xubuntu-Version mit Xfce-Desktop und VDR bereit. Weiterhin wird freeVDR ab Version 2.0 mit XBMC angeboten. Zusätzlich erfolgt die Installation über einen eigenen Installer. [58]

Hauptaugenmerk der Entwickler liegt mit dieser Distributionsversion 3.0 unter anderem auf dem VDPAU [siehe Glossar] -Support.

Zusätzlich sollte erwähnt werden, dass die im System installierte Version eine Betaversion darstellt. Verwendet wird hier die Distribution in der Version 3.0. Aufgrund der damit verbundenen Weiterentwicklung kann eine Auflistung integrierter Softwarekomponenten und Softwareversionen sehr schnell veralten.

In Anbetracht des Entwicklungsstandes zum Installationszeitpunkt baut Version 3.0 auf Ubuntu lucid 10.04 LTS auf. Der neu aufgesetzte Installer unterstützt eine manuelle sowie eine automatische Partitionierung.

Nach erfolgreicher Installation, dem Einrichten von Grafiktreiber und DVB-Treiber, steht dem Nutzer ein voll funktionsfähiges Media Center zur Verfügung. Ausgehend von den

58 vgl. http://www.vdr-wiki.de/wiki/index.php/FreeVDR, Stand: 07.12.2010

Grundfunktionen kann der Nutzer zwischen drei Bestandteilen in freeVDR wählen. Unter der grafischen Arbeitsplatzumgebung Xfce4 können Einstellungen rund um freeVDR durch ein Admin-Tool vorgenommen und nachträglich Programme installiert werden. Anhand des VDR kann über die DVB-Karte Live-TV empfangen und aufgezeichnet werden. Zu guter Letzt steht dem Nutzer mit XBMC ein multimediales Anwendungsprogramm bereit. Ein einfaches Umschalten zwischen den Bestandteilen ist dank des freeVDR-Daemon [siehe Glossar] möglich. Kurze Umschaltzeiten kennzeichnen den freeVDR-Daemon gegenüber anderen Daemons in früheren freeVDR-Versionen [59].

Ausgehend einer nutzerfreundlichen und bequemen Steuerung ist LIRC bereits im System vorinstalliert. LIRC steht für Linux Infrared Remote Control und erlaubt das Dekodieren und Senden von Signalen mittels Infrarot über nahezu jede handelsübliche Fernbedienung. Lediglich ein LIRC-fähiges Empfangsmodul ist nötig [60]. Somit muss LIRC unter freevdr 3.0 nur noch eingerichtet werden.

In den folgenden Kapitelabschnitten erfolgt eine Beschreibung der Bestandteile von freeVDR.

4.5.2.1 XBMC Media Center

Eine ausführliche Vorstellung des Media Center's erfolgte zuvor im Abschnitt 3.2.

Wie bereits erwähnt, beinhaltet freeVDR ab Version 2.0 das XBMC Media Center. Allerdings wird XBMC standardmäßig ohne eine Option auf Live-TV angeboten. Diesen Punkt griffen Privatentwickler auf und statteten auf experimenteller Basis XBMC-Pakete mit VDR-Unterstützung als PVR-Backend aus. Genau solch ein Paket kommt in freeVDR 3.0 zum Einsatz. Jedoch kommt es durch ständige Weiterentwicklungen aufgrund von Fehlerbehebungen oder ähnlichen Dingen schnell zur Veraltung. Abhilfe schafft eine Aktualisierung des Repository [siehe Glossar].

Da die Nutzung des neuen Add-on-System's in XBMC im Vordergrund steht, war eine Aktualisierung unabdingbar. In der aktuellen Version werden die Pakete von Henning Pingel's (Hepis) Personal Package Archive (PPA) verwendet. PPA heißt übersetzt eigenes Paketarchiv und ist ein spezielles Software-Repository. Hepi's Archiv enthält experimentelle und oft ungetestete XBMC-Konstrukte (XBMC builds), basierend auf dem SVN branch pvr-testing2-Zweig [61]. Die Konstrukte werden mit PVR-Support angeboten. Fast wöchentliche Aktualisierungen kennzeichnen die Pakete des Entwickler's.

---

59 vgl. http://vdr-free.de/wiki/index.php?title=FreeVDR_3.0, Stand: 07.12.2010

60 vgl. http://www.vdr-wiki.de/wiki/index.php/LIRC, Stand: 08.12.2010

61 vgl. https://launchpad.net/~henningpingel/+archive/xbmc, Stand: 08.12.2010

4.5.2.1.1 Add-on-System in XBMC

Seit einiger Zeit wird von den Entwicklern am neuen Add-on-System für XBMC entwickelt. Mit Einführung von XBMC 10.0 Dharma soll es fester Bestandteil im Media Center werden. Laut den Entwicklern soll es voll-ausgereift mächtiger als das alte SVN Repo-System sein. Über das Software-Repository konnten in früheren Versionen nachträglich Erweiterungen in XBMC installiert werden. Doch nicht nur in Beta-Versionen von Dharma ist das neue System bereits integriert. Auch in der hier eingesetzten PVR-Version von Henning Pingel kommt diese Neuentwicklung zum Einsatz. Deswegen sind einige Erläuterungen des System's betreffend erforderlich.

Grundlegende Erläuterungen zu Add-ons wurden schon im Kapitelabschnitt 3.2.4 niedergeschrieben. Hinsichtlich der Ordnerstruktur und dazugehöriger Dateien sind wichtige Merkmale für ein erfolgreiches Einbinden in XBMC einzuhalten.

Jedes Add-on ist in einem eigenen Ordner unter einer spezifischen Ordnerstruktur gespeichert und wird über eine XML-Datei beschrieben. Der Ordner muss sich unter */root/.xbmc/addons* befinden. Die XML-Datei namens addon.xml muss im Add-on-Ordner enthalten sein. Darüber hinaus können noch andere Dateien optional im Ordner hinterlegt sein. Entscheidet sich z.B. der Nutzer für ein angezeigtes Add-on-Icon, muss eine Datei namens icon.png im Ordner vorhanden sein. Die Datei muss im PNG-Format vorliegen und bestimmte Eigenschaften wie die Bildgröße einhalten. Alle diese Dateien müssen im Wurzelverzeichnis des Add-on's zu finden sein. Zusätzlich können Dateien mit Übersetzungen oder ähnlichen in einem Unterordner hinterlegt werden.

Wichtigste Datei neben den eigentlichen Add-on im Ordner stellt die addon.xml-Datei dar. Diese Datei beschreibt die Art der Erweiterung, welches das Add-on bietet. Ebenso werden für die Interaktion mit XBMC benötigte Schnittstellen beschrieben. Nachfolgend soll auf die wichtigsten Strukturelemente der XML-Datei eingegangen werden:

```
<?xml version="1.0" encoding="UTF-8"?>
<addon
    ...
    <requires>
        ...
    </requires>
    <extension
        ...
    <extension point="xbmc.addon.metadata">
        <summary>...</summary>
        <description>...</description>
        <disclaimer>...</disclaimer>
        <platform>...</platform>
    </extension>
</addon>
```

Nach der XML-typischen Kopfzeile folgt das *addon*-Element in Zeile 2. Das Element muss vorhanden sein und präsentiert die Daten über das Add-on-Paket als Ganzes. Innerhalb des *addon*-Element's befindet sich das *requires*-Element. Dort werden die Anforderungen für die Funktionalität des Add-on's beschrieben. Anschließend an das *requires*-Element kann es ein oder mehrere *extension*-Elemente geben. Jedes Element beschreibt einen Teil des Add-on's. Schließlich enthält die Struktur ein spezielles *extension point*-Element. In dem Element wird das Add-on für den Anwender beschrieben [62].

Damit ist das Add-on ausreichend beschrieben und kann in XBMC eingebunden werden. Nur so wird das Add-on folglich in der Ordnerstruktur von XBMC angezeigt. Andernfalls kann von einer fehlerhaften Implementierung der addon.xml-Datei ausgegangen werden. Auf die genauen Inhalte der Strukturelemente wird im Kapitel 5 eingegangen.

4.5.2.1.2 Built-In-Funktionen

Während die mitgelieferten Module xbmc und xbmcgui die Kontrolle über die Oberfläche garantieren, ermöglichen die Built-In-Funktionen die Kontrolle über andere Programmteile in XBMC. Mit dem Aufruf *executebuiltin* können Kommandos zur Funktionskontrolle vieler Software-Komponenten realisiert werden.

4.5.2.2 Xfce4-Desktopumgebung

"Xfce is a lightweight desktop environment for various *NIX systems. Designed for productivity, it loads and executes applications fast, while conserving system resources." - Olivier Fourdan, creator of Xfce. [63]

Mit diesen Worten wird treffenderweise die Desktopumgebung Xfce von seinem Entwickler auf der Anbieterwebseite beschrieben. In der Tat ist die grafische Arbeitsumgebung auf Produktivität optimiert. Sie kann Anwendungen schnell laden und trotzdem die Systemressourcen schonen. Der modulare Aufbau, bestehend aus einer Vielzahl von Einzelkomponenten, ermöglicht dem Nutzer die für sich ideale Arbeitsplatzumgebung zu schaffen.

Sehr benutzerfreundlich gestaltet sich die Administration mit dem vorinstallierten freeVDR-Admin-Tool. Das webbasierte Tool erleichtert die Administration und Konfiguration von freeVDR mit all seinen Komponenten. Über das Tool lässt sich z.B. das zuerst gestartete Programm nach dem Systemstart festlegen. Hierfür bietet sich im Anwendungsfall XBMC als Startprogramm an. Ebenfalls können über das Tool Grafik- und Netzwerk-Einstellungen vorgenommen werden. Ein Überblick installierter Plugins, aber auch das Hinzufügen weiterer, ist unter dem speziellen Reiter möglich. Zur Fehlersuche wird dem Nutzer eine Log-Datei angeboten.

---

62 vgl. http://wiki.xbmc.org/index.php?title=Add-ons_for_XBMC_%28Developement%29, Stand: 14.12.2010

63 vgl. http://www.xfce.org/, Stand: 08.12.2010

4.5.2.3 VDR mit DVB-T USB-Stick und vorinstallierten Plugin's

VDR ist eine Linux-Software zur Nutzung des Computer's als digitalen Videorecorder. Mit einem VDR lassen sich die Möglichkeiten von aktiven DVB-Karten nahezu uneingeschränkt nutzen. Das Videosignal wird hierbei direkt von der DVB-Karte bezogen. Neben den bekannten Funktionen von digitalen Fernsehen wie EPG, Kanalliste etc. lassen sich mit Hilfe von Plugins etliche Erweiterungen integrieren. So können z.B. MP3-Player oder DVD-Player in den Funktionsumfang aufgenommen werden. [64]

Mit freeVDR 3.0 wird der VDR in der Version 1.7.14 installiert. Voraussetzung für den Gebrauch ist eine digitale TV-Karte. In diesem System erfolgt der Empfang von Live-TV über einen DVB-T USB-Stick. Die genaue Bezeichnung der Karte lautet Terratec Cinergy DT USB XS Diversity. Der Stick verfügt über zwei DVB-T Tuner. Somit kann z.B. parallel neben Fernsehen eine Sendung aufgezeichnet werden.

Über zwei vorinstallierte Frontendprogramme erfolgt die Bildwiedergabe über den VDR. VDR-sxfe ist ein Frontendprogramm und Bestandteil des vorinstallierten Xineliboutput-Plugin. Allerdings läuft es nur in Verbindung mit dem X-Server. Das VDR-sxfe-Frontend wird auch noch als X11 [siehe Glossar] -Frontend bezeichnet. Xineliboutput unterstützt ebenfalls lokale und entfernte Frontends. Für die Ansteuerung entfernter Frontends generiert das Plugin einen HTTP-Stream auf Port 37890 [65]. Als zweites Frontendprogramm wird das Xine-Plugin mit Xine-ui mitgeliefert. Entwickelt wurde Xine für Budget-DVB-Karten. Dabei greift Xine direkt den MPEG-2 TS von der DVB-Karte ab. Die anschließende Dekodierung und das Demuxen erfolgt softwareseitig durch geeignete Medienspieler wie dem Xine-ui [66].

Ein weiteres Plugin wird bereits mit Installation von freeVDR angeboten. Mit Hilfe des Streamdev-Plugin kann der Nutzer Fernsehkanäle über den Streamdev-Server an Clients im Netzwerk streamen. Wie bereits erwähnt besteht das Plugin aus Server und Client. Bei einer VDR-zu-VDR-Lösung dient der Streamdev-Client als Eingabegerät und ist mit Softwareplayern wie dem Xine-Player koppelbar. So kann auf Seiten des Client's der VDR ohne DVB-Karte betrieben werden. Andernfalls erfolgt der Stream direkt auf einen Softwareplayer wie VLC oder dem Windows Media Player. Vorteilhaft gegenüber dem Xinelibouput-Plugin ist hier die Unabhängigkeit zwischen den Clients. Während beim Streamdev-Plugin alle unabhängig voneinander agieren können teilen sich beim Xineliboutput-Plugin die Clients den gleichen VDR. Alle Clients sehen immer nur das gleiche Programm. Durch den Dual-Tuner der DVB-T-Karte können somit zwei Clients

64 vgl. http://de.wikipedia.org/wiki/Video_Disk_Recorder, Stand: 09.12.2010

65 vgl. http://www.vdr-wiki.de/wiki/index.php/Xineliboutput-plugin, Stand: 09.12.2010

66 vgl. http://www.vdr-wiki.de/wiki/index.php/Xine-plugin, Stand: 09.12.2010

unterschiedliche Sendungen konsumieren. Auch hier erfolgt das Streamen mittels eines HTTP-Streams zum Client. [67]

4.5.3 Software des CTI-Server's

4.5.3.1 TapiPhone

TapiPhone stand dem Entwickler bereits als funktionsfähige Software zur Verfügung. Geschrieben ist das Programm in Visual C++. Es erfolgte, für die Kommunikation mit XBMC, die Implementierung eines Server-Socket und Client-Socket in das bestehende Programm.

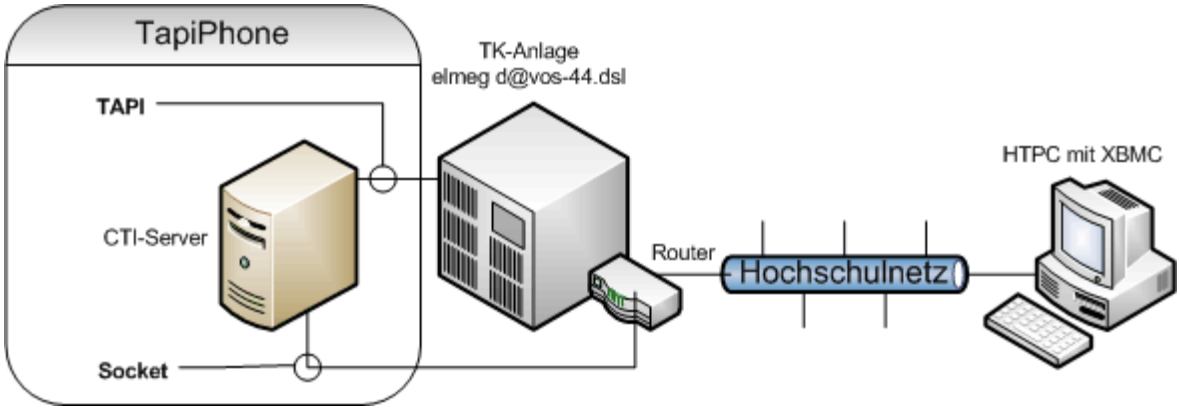

Abbildung 4-15: Schnittstellen von TapiPhone

Über den Aufruf von TAPI-Funktionen durch die TAPI.dll erfolgt der Nachrichtenaustausch zwischen der TK-Anlage und dem CTI-Server. Zuständig für die Zusendung der Nachrichten ist der TAPI-Treiber. Empfangen und ausgewertet werden die Nachrichten in der CTI-Anwendung innerhalb der Callback-Funktion. Damit die Nachrichten durch die CTI-Anwendung empfangen und ausgewertet werden können, muss bei Initialisierung einer TAPI-Session die Adresse der Callback-Funktion angegeben werden. Nachrichten enthalten Informationen über den Zustand des Endgerätes oder den durchgeführten Aktionen wie MakeCall oder DropCall.

TapiPhone repräsentiert diese CTI-Anwendung. Die Software fängt ausgetauschte Nachrichten ab und wertet diese in der Callback-Funktion innerhalb des Programm's aus.

Implementierte Socket's übernehmen als Kommunikationsendpunkte die Verbindung zu XBMC über das Hochschulnetzwerk in die andere Richtung. Ausgewertete Nachrichten in der Callback-Funktion erfahren in Abhängigkeit ihres Inhaltes und Bedeutung in den Socket-Methoden bestimmte Behandlungen. Anschließend werden aufbauend auf den Behandlungen eigens definierte Nachrichten über den Socket an den Server oder Client übertragen.

---

67 vgl. http://www.vdr-wiki.de/wiki/index.php/Streaming, Stand: 13.12.2010

Zum Übertragen der Rufnummer und eigens definierter Nachrichten bei einen ankommenden Anruf ist der Client-Socket in die Callback-Funktion implementiert. Dieser wird beim Anrufeingang initialisiert und mit dem Server-Socket auf Seiten des HTPC verbunden. Das heißt, dass bei Anrufeingang die vom TAPI-Treiber gesendete Rufnummer anhand der *GetPhoneNumber()*-Funktion ermittelt und anschließend über den Socket zum Skript *popup.py* in XBMC zur Anrufsignalisierung gesendet wird. Innerhalb der Callback-Funktion wird die *GetPhoneNumber()*-Funktion bei einen Anrufereignis aufgerufen. Anschließend wird der Socket noch für den Versand von Statusnachrichten bei Abheben und Auflegen des Hörer's am ISDN-Telefon verwendet. Erst beim Auflegen des Hörer's wird der Socket clientseitig geschlossen.

Der integrierte Server-Socket empfängt von der Client-Anwendung eigens definierte Steuerbenachrichtigungen für die TAPI-Funktionen MakeCall und DropCall. Initialisiert wird der Socket beim Start der CTI-Anwendung. In seinen Ereignismethoden werden außerdem Behandlungen zur Unterscheidung der TAPI-Funktion und zur Erkennung der zu wählenden Rufnummer ausgeführt.

4.5.3.2 TAPI-Treiber und TAPI Configurator

Neben der eigentlichen TK-Anlage liefert elmeg Treiber und Programme zur Einrichtung der TK-Anlage mit. Der mitgelieferte TAPI-Treiber ermöglicht die Anbindung an ein Programm für computerunterstützte Telefonieanwendungen.

Mit dem TAPI Configurator wird die TAPI-Schnittstelle konfiguriert. Anhand des Konfigurator's kann der TAPI-Treiber an ein Programm, das den Treiber verwendet, angepasst werden. Voraussetzung ist die Erstkonfiguration der TK-Anlage und die Verbindung dieser mit dem Computer. Danach kann sowohl der Leitungsname als auch die Nebenstellen, die zur Verwendung von CTI zum Einsatz kommen sollen, ausgewählt werden.

Im Systemkonzept überwacht der TAPI-Treiber das an der TK-Anlage angeschlossene ISDN-Telefon. Daher musste die intern zugewiesene Rufnummer des ISDN-Telefon's dem TAPI Configurator übergeben werden.

4.5.3.3 Professional Configurator

Einstellungen zu Leitungsmerkmalen der TK-Anlage erfolgt über den Professional Configurator. So können externe Mehrfachrufnummern (MSN) eingerichtet und an einzelne interne Teilnehmer oder einem Team aus mehreren Teilnehmern zugewiesen werden. Darüber hinaus können separat für jeden einzelnen Teilnehmer besondere Leistungsmerkmale wie Berechtigungen, Amtsholung und vieles mehr eingerichtet werden.

Notwendig war hier die Zuordnung der primären ISDN-Rufnummer an das vorgesehene Endgerät. So werden alle externen Anrufe dem ISDN-Telefon zugeordnet. Es erfolgt nur dort eine Anrufsignalisierung.

Unter dem Menüpunkt *Netzwerk* sind unter anderem Einstellungen zum Internetzugang, des lokalen Netzwerk's, der IP-Adressvergabe und Filtereinstellungen vorzunehmen. Da besonders hier wichtige Einstellungen vorzunehmen waren, soll eine detailliertere Beschreibung ausgewählter Unterpunkte erfolgen.

Über den Netzwerk-Unterpunkt *Internetzugang* wird der gemeinsame Zugang aller angeschlossen Computer in das Internet konfiguriert. *Router/LAN* ermöglicht eine Zuweisung von IP-Adresse und Subnetzmaske der TK-Anlage. Mittels der internen IP-Adresse wir der Fernzugang zur Anlage über einen Ethernet-Port ermöglicht. Zum Einrichten des integrierten DHCP [siehe Glossar] -Server's dient der Untermenüpunkt *Adresszuordnung*. Hier kann bei automatischer Adressvergabe die Startadresse und diverse Subnetzparameter eingestellt werden.

Die Auswahl *Filter* ermöglicht für bestimmte IP-Adressen im lokalen Netzwerk einen oder mehrere Ports für den ankommenden und ausgehenden Datenverkehr zu sperren beziehungsweise zu öffnen. Für das Systemkonzept war hier ein sogenanntes Portmap notwendig. Damit der CTI-Server über das Hochschulnetz angesprochen werden kann, ist der vom Server-Socket in TapiPhone verwendete Port in der TK-Anlage zum CTI-Server durchzuschalten (mappen). Ansonsten wird jeglicher ankommender Datenverkehr durch die Anlage blockiert.

4.5.4 Überblick der verteilten Kommunikationsendpunkte

Aufgrund der unterschiedlichen Aufgabenbereiche innerhalb der Anwendungen und dem damit verbundenen Nachrichtenaustausch wurde für das Systemkonzept eine Aufteilung von Client- und Server-Socket vorgenommen. Diese Tatsache wurde schon hinreichend in vorherigen Kapitelabschnitten erläutert Die unten stehende Abbildung 4-16 zeigt die Kommunikationsendpunkte in den jeweiligen Anwendungen und der Rolle um Client-Server-Modell. Sie soll für einen bessere Verständlichkeit dienen.

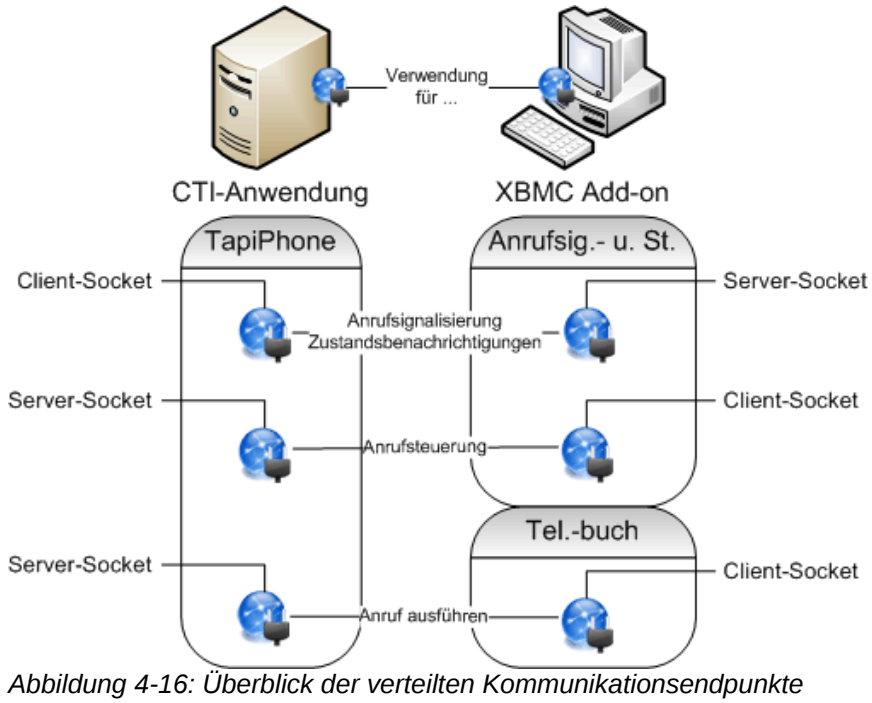

Abbildung 4-16: Überblick der verteilten Kommunikationsendpunkte

5 Implementierung

Im folgenden Kapitel soll die Implementierung der Anforderungen an das Systemkonzept vorgestellt werden. Es erfolgen detaillierte Erläuterungen zu einzelnen Schritten, die für den Aufbau des CTI-Server's und des Media Center's mit all ihren Komponenten durchzuführen waren. Diese Schritte umfassen Ausführungen von der Installation der Hard- und Software bis zum Test der gewünschten Anforderungen. Die Erläuterungen zu Quelltextabschnitten finden nur für relevante Abschnitte statt.

5.1 Einrichtung des CTI-Server's

Der Weg zu einen funktionsfähigen CTI-Server im Systemkonzept gliedert sich in Teilschritte. Nach der Installation von Windows XP mit Service Pack 3 erfolgt der Anschluss der TK-Anlage an den Server. Nachdem alle notwendigen Treiber installiert und wichtige Netzwerkeinstellungen vorgenommen sind, kann der Anschluss an das Hochschulnetzwerk vorgenommen werden. Der letzte Schritt befasst sich mit der Erweiterung von TapiPhone.

Auf die Installation und Einrichtung von Windows XP mit allen relevanten Treibern soll hier nicht weiter eingegangen werden. Daher beginnt der Einstieg nach dem Anschluss der TK-Anlage zwischen dem Ethernet-Port und der Netzwerkkarte des CTI-Server's mittels eines CAT.5-Kabel. Zu erwähnen ist noch ein wichtiger Punkt in den Konfigurationseinstellungen der Netzwerkkarte des Server's. Hier muss bei den TCP/IP-Einstellungen unbedingt die automatische IP- und DNS-Adressvergabe eingestellt sein. Der Server bezieht die Adressen automatisch vom integrierten DHCP-Server in der TK-Anlage.

5.1.1 Installation und Konfiguration der TK-Software

Aus dem Benutzerhandbuch der elmeg TK-Anlage ist zu entnehmen, dass in der Grundeinstellung diese über die interne IP-Adresse 192.168.1.250 zu erreichen ist. Nach dem Einlegen der mitgelieferten elmeg Anwender-CD erscheint der Eröffnungsbildschirm des Autostart. Über den Menüpunkt *Kompakte Systeme* findet sich die elmeg D@vos-44.dsl-TK-Anlage. In Folge stehen dem Anwender mehrere Softwareprogramme als Bundle mit der Bezeichnung WIN-Tools zur Verfügung.

Abbildung 5-1: Startfenster der Softwareinstallation

Für die Einrichtung der TK-Anlage und TAPI wird sowohl der Professional Configurator als auch der TAPI Configurator benötigt. Diese beiden Programme sind in dem Bundle enthalten. Daher muss die Installation von WIN-Tools erfolgen. Notwendige TAPI Treiber werden automatisch mit installiert. Im Anschluss an die Installationsprozedur kann die Anlage eingerichtet werden.

5.1.1.1 Konfiguration der Anlage anhand des Professional Configurator

Die TK-Anlage wird mit speziellen Grundeinstellungen ausgeliefert. Ausgewählte Einstellungen können so für das Konzept übernommen werden. So werden interne Rufnummern in Nummernbereiche unterteilt. Für Geräte am internen ISDN-Bus sind die Rufnummern 20 bis 29 vergeben. Geräte an den analogen Anschlüssen beziehen standardmäßig die Rufnummern 10 bis 13. Ebenfalls sind Nummern für CAPI und dem DSL-Router reserviert. Jedoch spielen diese im Systemkonzept keine Rolle und sollen deswegen nicht weiter benannt werden. Auf Grundlage dessen sollen nun die Anschlüsse der S0-Schnittstelle und der analogen Anschlüsse konfiguriert werden.

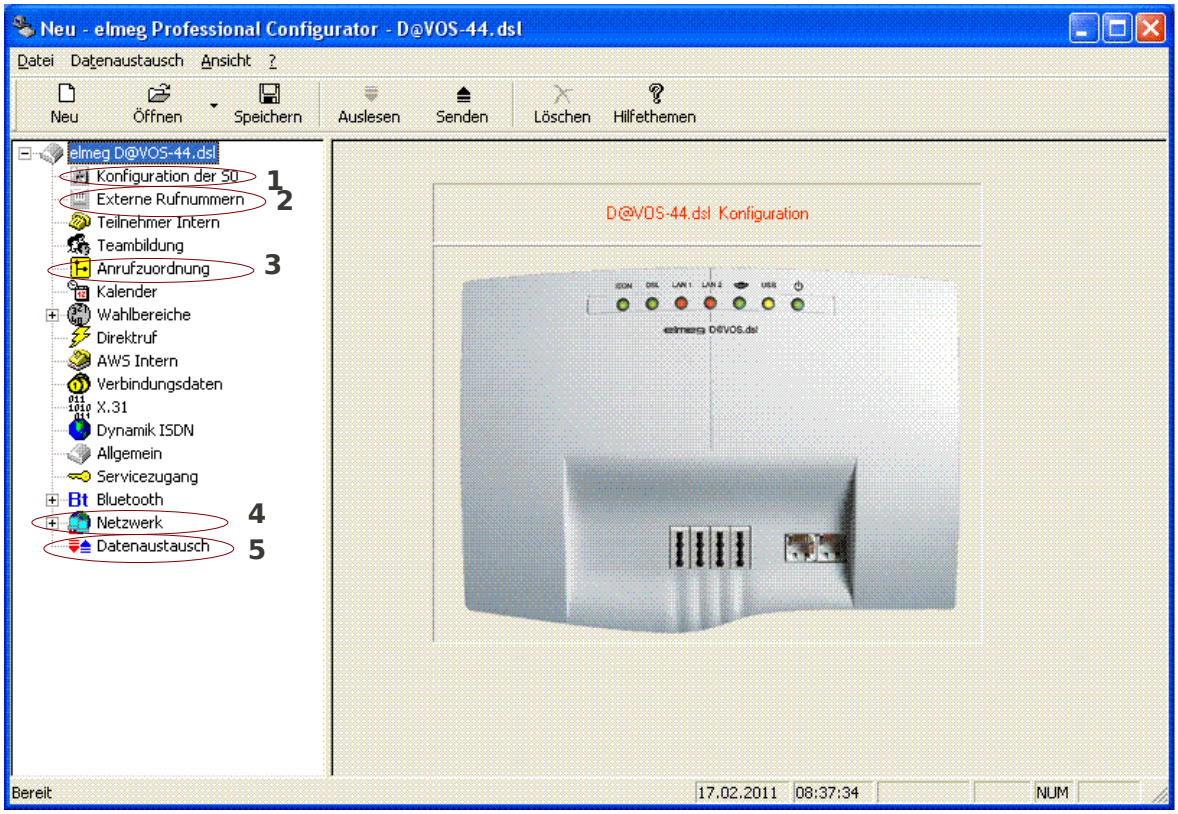

Abbildung 5-2: Professional Configurator der TK-Anlage

Im ersten Konfigurationsschritt wird unter *Konfiguration der S0* (siehe Abbildung 5-2, 1) dem Anschluss Basis S0-1 (siehe Abbildung 5-3, 1) die interne Rufnummer 20 (siehe Abbildung 5-3, 2) zugewiesen. Demnach ist das angeschlossene ISDN-Telefon unter der internen Rufnummer 20 zu erreichen. Der ausgewählte Anschluss ist für den internen Mehrgeräteanschluss fest eingestellt.

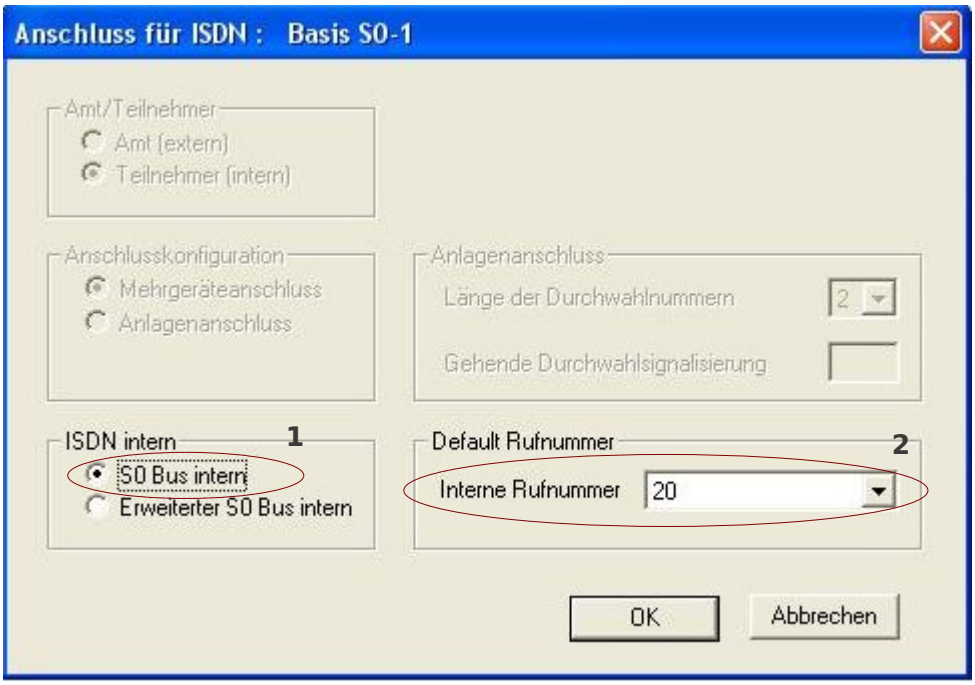

Abbildung 5-3: Einstellung der S0-Schnittstelle

Weiterhin ist der Anschluss Basis S0-2 für den externen Gebrauch als Mehrgeräteanschluss fest eingestellt. Allerdings sind Einstellungen für den Basis S0-2- oder auch dem Basis CAPI-Anschluss nicht vorzunehmen.

Beim nächsten Schritt werden unter *Externe Rufnummern* (siehe Abbildung 5-2, 2, Seite 67) die externen ISDN-Mehrfachrufnummern eingetragen. Eine primäre Rufnummer und zwei Nebenstellen-Rufnummern sind bei dem vorliegenden ISDN-Anschluss verfügbar.

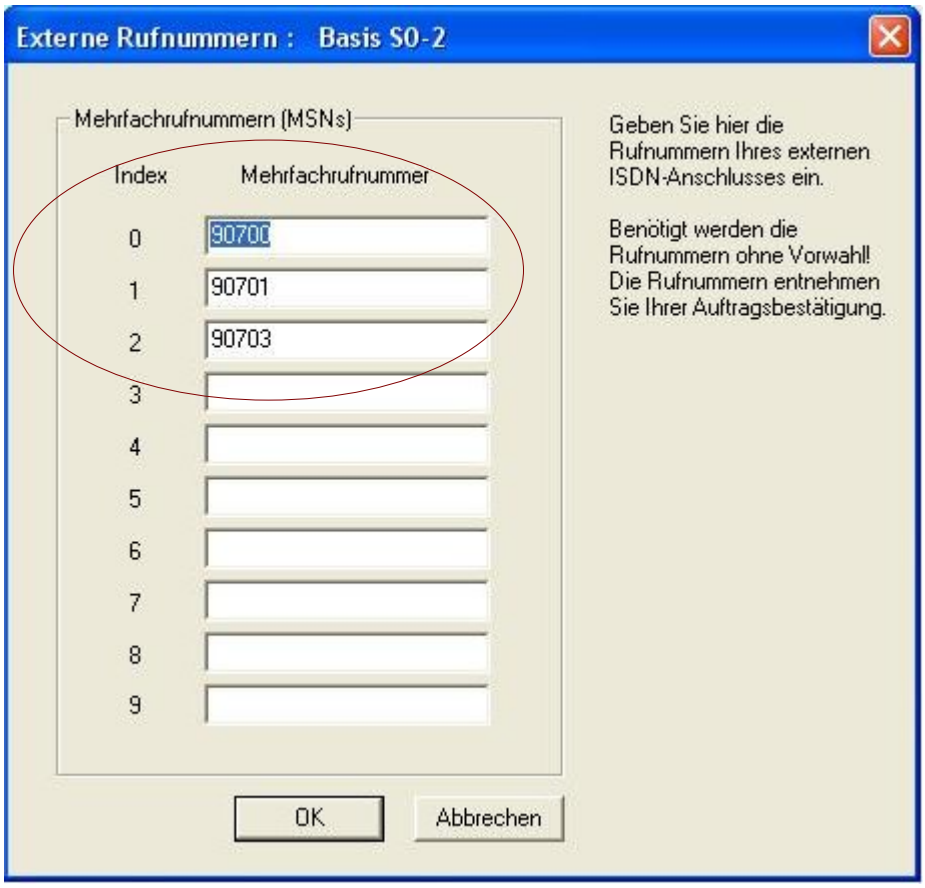

Abbildung 5-4: Zuweisung der Mehrfachrufnummern

Der ISDN-Anschluss im Labor für Kommunikationstechnik hat die Rufnummern 90700, 90701 und 90703 zugewiesen bekommen. Primäre Rufnummer ist die 90700 und alle ausgehenden Anrufe werden mit dieser Rufnummer signalisiert. Eine benötigte Rufnummernvervollständigung zeigt die Abbildung 5-4.

Unter den folgenden Menüpunkten *Teilnehmer Intern* und *Teambildung* können Leistungsmerkmale und Berechtigungen der einzelnen, internen Teilnehmer vergeben werden. Hier sind keine weiteren Einstellungen einzutragen.

Eine weitere Konfiguration ist unter *Anrufzuordnung* durchzuführen (siehe Abbildung 5-2, 3, Seite 67). Auf die externe Rufnummer 90700 wird die interne Rufnummer 20 abgebildet (siehe Abbildung 5-5, 1). Mit dieser Einstellung enden alle externen Anrufe auf dem Endgerät an der internen S0-Schnitstelle. Neben der Zuordnung einer einzelnen

Rufnummer besteht auch die Möglichkeit, einem Team eine externe Rufnummer zuzuordnen. Diese Option soll im Systemkonzept aber nicht zum Einsatz kommen.

Abbildung 5-5: Auswahlfenster der Anrufzuordnung

Abschließende Konfigurationseinstellungen sind jetzt noch unter *Netzwerk* vorzunehmen (siehe Abbildung 5-2, 4, Seite 67). Um die Anlage über das Hochschulnetz betreiben zu können, musste im Vorfeld eine Anmeldung dieser am Rechenzentrum mit Hilfe der MAC-Adresse vorgenommen werden. Erst danach kann die Anlage über hochschulinterne DHCP-Server alle relevanten Netzwerkeinstellungen beziehen.

Für die Anbindung an das Hochschulnetz und der damit inbegriffenen Verbindung zum Internet ist unter *Internetzugang* ein neuer Internet Service Provider (ISP) hinzuzufügen.

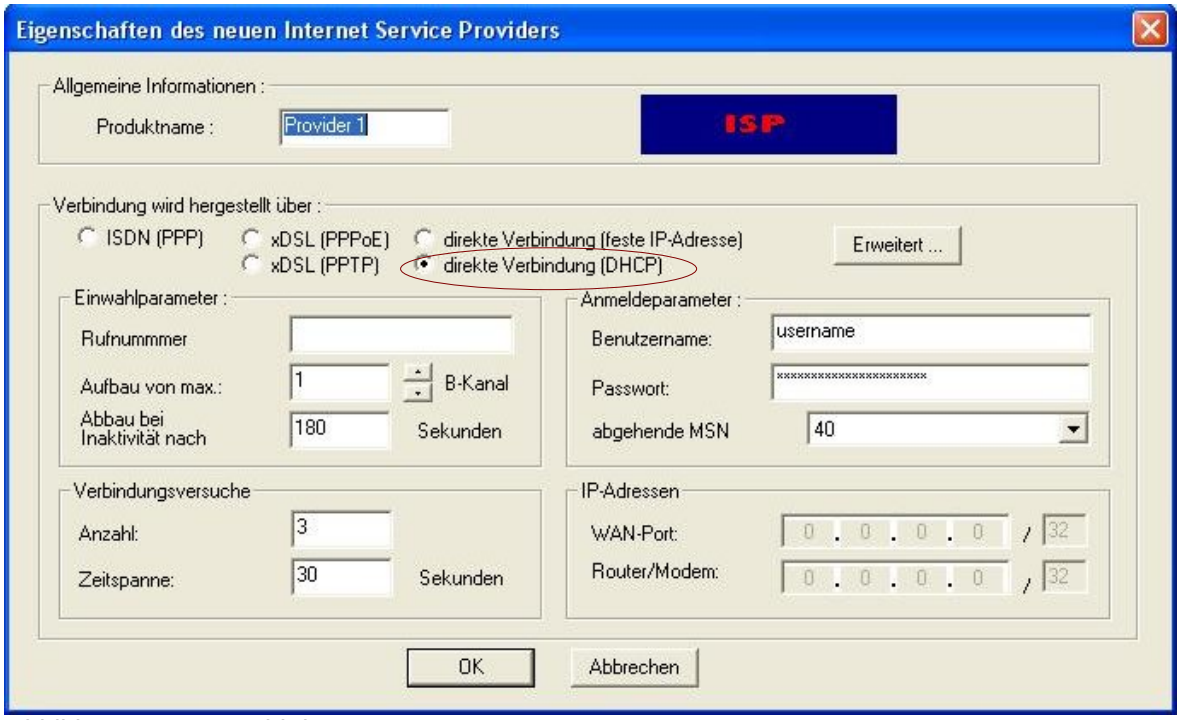

Abbildung 5-6: Auswahl des ISP

Mit der Auswahl d*irekte Verbindung (DHCP)* wird mit Anschluss eines CAT.5-Kabels an den WAN-Port der TK-Anlage die Verbindung mit dem Hochschulnetzwerk hergestellt und die benötigten Serveradressen bezogen (siehe Abbildung 5-6). Einwahl- und Anmeldeparameter sowie Einstellungen zu Verbindungen und IP-Adressen entfallen.

Anlagenparameter zur internen IP-Adresse und Subnetzmaske sind unter *Router/LAN* aufgelistet. Diese Parameter müssen allerdings nicht verändert werden. Wie bereits erwähnt ist der Anlage die IP-Adresse 192.168.1.250 und als Subnetzmaske 255.255.255.0 zugewiesen.

Der integrierte DHCP-Server der Anlage übernimmt die Adresszuordnung im LAN. Standardmäßig findet man unter *Adresszuordnung* als Startadresse für die dynamische Vergabe der IP-Adressen an Clients die 192.168.1.50. Dieser Parameter bedarf keinerlei Veränderungen. Auch die weiteren Parameter können so belassen werden.

Nachfolgend sind wichtige Einstellungen bei den IP-Filtern zu machen. Unter *Filter* muss der Port für den eingehenden Datenverkehr zum CTI-Server durchgeschaltet werden. Andernfalls werden die Datenpakete blockiert. Über den Button *Neu* wird eine neue Filterregel hinzugefügt. Hier können nun die gewünschten Einstellungen erfolgen (siehe Abbildung 5-7, 1, 2).

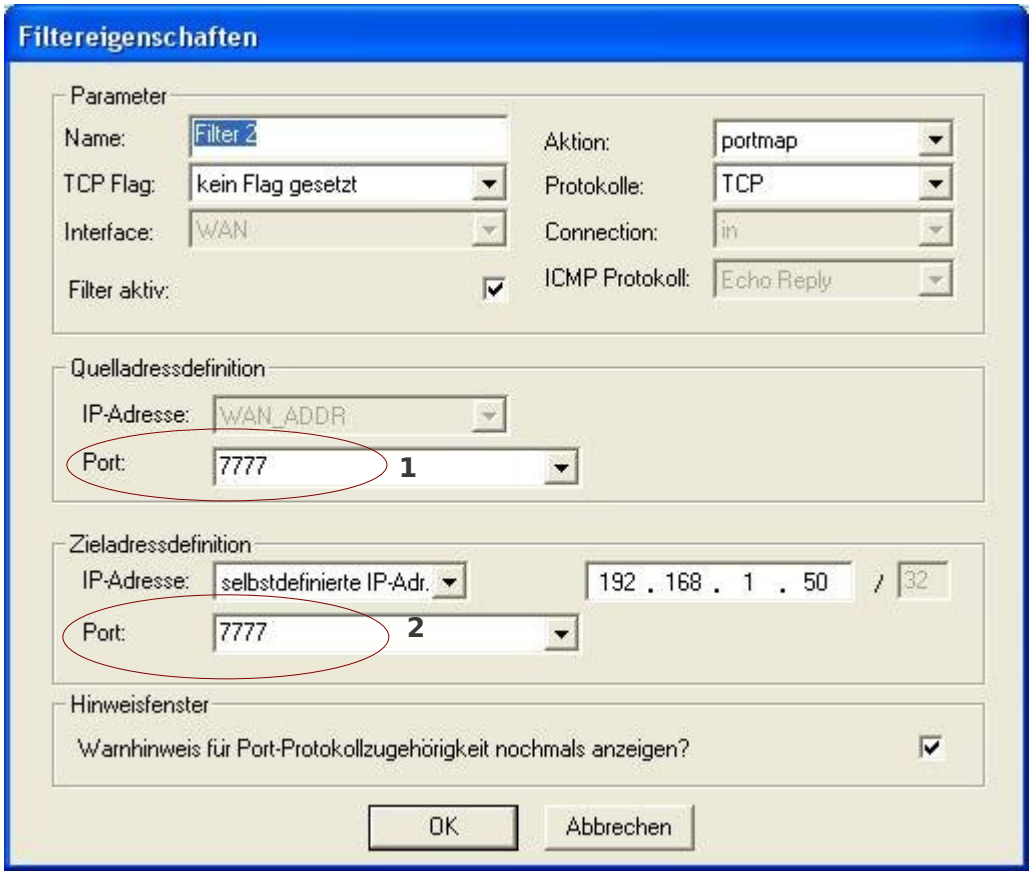

Abbildung 5-7: Einstellungen der Filterregel

Der Port 7777 wird in TapiPhone unter der Socket-Implementierung definiert. Darüber hört und wartet der Socket-Server auf Assoziationen des Client's auf Seiten der Add-ons in XBMC.

Letztendlich sind die Konfigurationsdaten zur Anlage zu Senden. Dafür müssen noch Einstellungen zur Schnittstelle zwischen der Anlage und dem Server unter *Datenaustausch* vorgenommen werden. Da die Verbindung über den Ethernet-Port hergestellt ist, muss der Punkt *LAN/USB* ausgewählt sein. Folglich muss die IP-Adresse der Anlage als Einstellungsparameter angegeben werden (siehe Abbildung 5-8, 1).

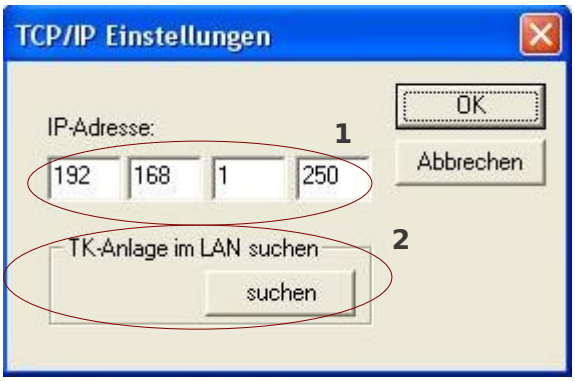

Abbildung 5-8: Einstellung zur Schnittstelle

Überprüft wird die Einstellung über den Button *TK-Anlage im LAN* suchen (siehe Abbildung 5-8, 2). Erst danach kann das Senden der Konfigurationsdaten durch drücken des Button *Konfiguration senden* erfolgen. Nun wird noch der vier-stellige PIN abgefragt. Standardmäßig lautet der PIN *0000* und ist bei Aufforderung einzutragen (siehe Abbildung 5-9).

Abbildung 5-9: Abfrage der PIN1

Jederzeit kann die PIN1 durch den Anwender geändert werden. Hierfür steht der Button *PIN1 ändern* unter *Datenaustausch* zur Verfügung.

Abschließend sollten die Einstellungen lokal oder auf einem externen Speichermedium gespeichert werden. Dies ist über *Datei → Speichern unter* möglich. Bei jeden Neustart des Professional Configurator ist das Laden der abgespeicherten Datei notwendig.

5.1.1.2 Konfiguration der TAPI-Schnittstelle

Mit der Installation des TAPI Configurator wird auch der TAPI-Treiber der TK-Anlage installiert. Im nächsten Schritt muss für die Telefonieanwendung eine in Frage kommende und zuvor eingerichtete Nebenstelle ausgewählt werden. Mögliche Nebenstellen setzen sich aus allen ISDN-Anschlüssen, analogen Anschlüssen, CAPI- und DSL-Router-Anschlüssen zusammen.

Nach Start des TAPI Configurator erscheint eine Liste aller Nebenstellen. Das Systemkonzept sieht für die Telefonieanwendung das am ISDN-Bus angeschlossene Endgerät vor. Demnach ist das Häkchen für die Nebenstelle 20 zu setzen (siehe Abbildung 5-10).

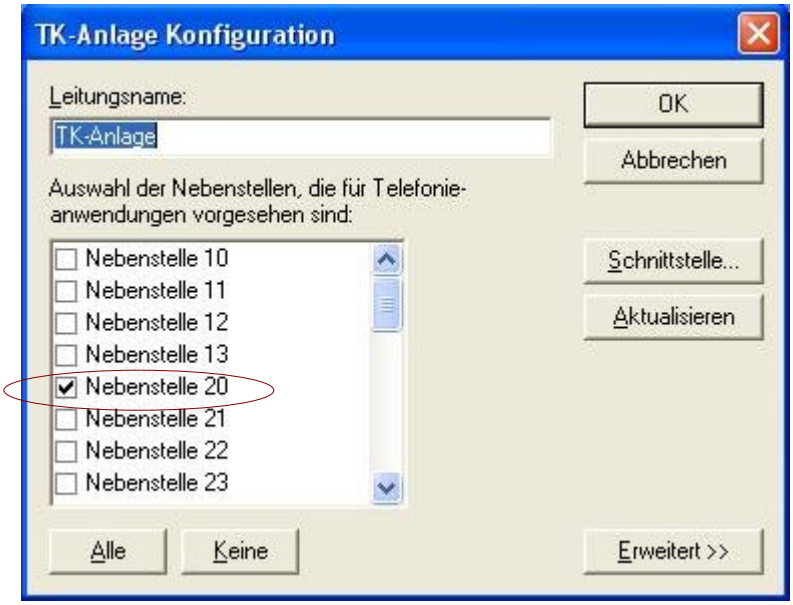

Abbildung 5-10: Auswahl der Nebenstelle im TAPI-Configurator

Optional kann der Anwender einen anderen Leitungsnamen vergeben.

Nachfolgend sollte noch einmal eine Überprüfung der Schnittstelle unter dem gleichnamigen Button erfolgen.

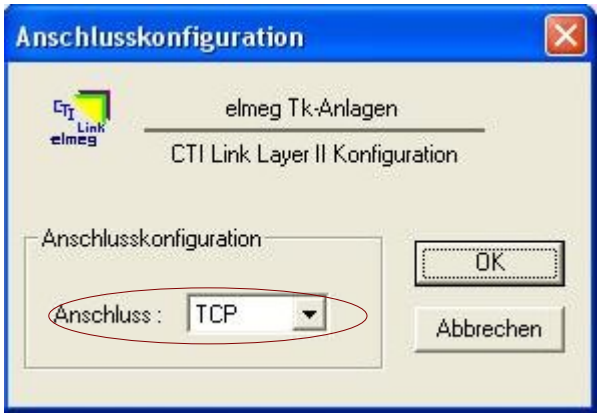

Abbildung 5-11: CTI-Link Konfiguration

Da die Verbindung über Ethernet realisiert wird, muss der Anschluss auf *TCP* eingestellt sein (siehe Abbildung 5-11).

Nach Bestätigung der Einstellungen öffnet sich automatisch ein Fenster für *Telefon und Modemoptionen*. Das standardmäßig unter der Windows-Systemsteuerung zu findende Auswahlfenster konfiguriert Wahlregeln für das Telefonieren. An dieser Stelle muss für den eigenen Standort die Ortskennzahl eingetragen werden. Im Falle von Mittweida lautet die Ortskennzahl also 3727.

5.1.2 Erweiterung von TapiPhone

Gemäß den Anforderungen an die CTI-Anwendung erfolgt in TapiPhone die Implementierung eines Client- beziehungsweise Server-Socket und in Abhängigkeit von empfangenen Nachrichten die ausgewählten Ereignisbehandlungen.

In diesem Kapitelabschnitt sollen nur Erläuterungen zu Quelltextabschnitten erfolgen, die die Funktionalität der CTI-Anwendung verdeutlichen. Benötigte Headerdateien oder andere programmiertechnische Aspekte sollen hier nicht erläutert und untersucht werden. Unterteilt wird das Kapitel in die Dateinamen, die diesem Projekt angehören und für die Funktionalität von Bedeutung sind.

5.1.2.1 TapiPhoneClg.cpp

TapiPhoneClg.cpp stellt als Implementierungsdatei Funktionen von der Initialisierung bis zur Verbindung des Client- und Server-Socket sowie die Callback-Funktion bereit. Funktionen für die CTI-Funktionalität werden hier ebenfalls definiert.

Natürlich müssen die Socket's zuerst eingerichtet werden.

```
1     //Client- und Server-Socket einrichten
2     //Aufruf des Konstruktors
3     sendSocket = new CMyAsyncSocket(this);
4     Server-Socket = new MyAsyncServer-Socket(this);
5     //Erstellen des Window-Server-Socket
6     Server-Socket->Create(7777,1,FD_READ | FD_WRITE | FD_OOB | FD_ACCEPT |
FD_CONNECT | FD_CLOSE,(CString)"192.168.1.50");
7     //auf Clientverbindungen warten
8     Server-Socket->Listen();
```

Dafür wird in den Zeilen 3 und 4 der Konstruktor des Client-Socket's (Zeile 3) und des Server-Socket's (Zeile 4) erzeugt. Während die weiteren Socket-Funktionen des Client-Socket's erst in der Ereignisbehandlung der Callback-Funktion abgearbeitet werden, wird der Server-Socket bereits hier mit der Member-Funktion *Create()* erstellt (Zeile 6). Der Funktion werden Parameter für den Port, dem Socket-Typ, eine Bitmaske für bereitgestellte Netzwerkereignisse und die zu überwachende IP-Adresse übergeben. In Folge dessen wartet der Server auf Assoziationen (Zeile 8).

```
9 void CTapiPhoneDlg::CallbackFunction(DWORD hDevice,DWORD dwMsg,DWORD
dwParam1,DWORD dwParam2,DWORD dwParam3)
10 {Clientsocket
```

Die vom TAPI-Treiber gesendeten Nachrichten an die Anwendung werden in der Callback-Funktion ausgewertet und einer entsprechenden Ereignisbehandlung unterzogen (Zeile 9). Gesendete Nachrichten enthalten hier einen Anrufzustand (Call-Zustand). Call-Zustände sind z.B. Ruf angenommen, Ruf beendet, Rufnummer wird

gewählt usw. Der übergebene Parameter *hDevice* signalisiert das Handle auf den Call. Mit *dwMsg* wird der Wert LINECALL_STATE übergeben. Den neuen Call-Zustand präsentiert *dwParam1* und kann Call-Zustände in Form von Konstanten annehmen.

```
11    const char disc[] = "disconnect";
12    const char conn[] = "connect";
13    const char unkn[] = "unknown";
14    const char idle[] = "onhook";
```

Am Anfang werden die Nachrichten , die für die jeweiligen Ereignisse über den Socket gesendet werden sollen, als konstanter Datentyp char deklariert (Zeile 11 bis 14).

```
15    switch(dwMsg){
16    case LINE_CALLSTATE:
```

In der ersten switch-Anweisung wird der Wert von *dwMsg* ausgewertet (Zeile 15). Enthält der Parameter den Wert LINE_CALLSTATE, folgt der Anweisungsblock unter dem case-Label (Zeile 16). Darauf hin wird im nächsten switch-Anweisungsblock der neue Call-Zustand in *dwParam1* ausgewertet (Zeile 17).

```
17              switch(dwParam1){
18              case LINECALLSTATE_ACCEPTED:
19                      //Ruf wurde angenommen
20                      m_CallInfoStrg.InsertString(-1,strZeit+(CString)": Freizeichen erkannt");
21                      break;
22              case LINECALLSTATE_RINGBACK:
23                      // Freizeichen wurde erkannt
24                      m_CallInfoStrg.InsertString(-1,strZeit+(CString)": Die gewählte Nummer
wird gerufen");
25                      break;
26              case LINECALLSTATE_BUSY:
27                      // Besetztzeichen wurde erkannt
28                      m_CallInfoStrg.InsertString(-1,strZeit+(CString)" Nummer ist besetzt");
29                      break;
```

Die Call-Zustandskonstanten LINECALLSTATE_ACCEPTED (Zeile 18), LINECALLSTATE_RINGBACK (Zeile 22) und LINECALLSTATE_BUSY (Zeile 26) werden zwar in der Callback-Funktion ausgewertet, finden jedoch nur intern in der Anwendung ihre Aufgabe. Demzufolge dienen die dahinter liegenden Zustände keiner Signalisierung an die Server-Anwendung im Media Center.

```
30              case LINECALLSTATE_OFFERING:
31                      //Socket verbinden
32                      sendSocket->Create();
33                      sendSocket->Connect((CString)"141.55.243.248", 6666);
34                      //eingehender Call
```

```
35                      m_hCall=(HCALL)hDevice;
36                      //Rufnummer ermitteln
37                      number=GetPhoneNumber(m_hCall);
38                      //Rufnummer mitgesendet
39                      if(number!="") {
40                              m_CallInfoStrg.InsertString(-1,strZeit+((CString)": Anruf von ")
+m_pPhoneBook->m_vorname+((CString)" ")+m_pPhoneBook->m_nachname);
41                              char str[100];
42                              //Funktion zur Datentypumwandlung
43                              CT2CA _number(number);
44                              strcpy(str,_number);
45                              //Rufnummer wird gesendet
46                              sendSocket->sendData(str); }
47                      //Rufnummer unterdrückt
48                      else {
49                              m_CallInfoStrg.InsertString(-1,strZeit+((CString)": Anruf von ")
+number);
50                              //unknown wird gesendet
51                              sendSocket->sendData(unkn); }
52                      break;
```

Mit der Zustandskonstante LINECALLSTATE_OFFERING wird die erste Benachrichtigung für den Eintritt in den Zustand über den Socket an die Server-Anwendung gesendet (Zeile 30). Der Zustand beschreibt die Einleitung des Call's und der gleichzeitigen Signalisierung am Zielendgerät. Innerhalb des Anweisungsblock wird der Client-Socket über die IP-Adresse und dem Port des Server-Socket's verbunden. Das Handle des Call's wird als ganze Zahl größer Null dem Objekt *m_hCall* überschrieben (Zeile 35). Damit ist der Call eindeutig gekennzeichnet und kann für die Ermittlung der Rufnummer verwendet werden. Dafür ist eigens eine Funktion *GetPhoneNumber()* definiert (Zeile 37). Unterdrückt der Anrufer seine Rufnummer, tritt die erste Bedingung nicht ein und der Konstruktormethode *sendData()* wird ein *unknown* übergeben (Zeile 39, 48). Andernfalls erfolgt der Anweisungsblock ab Zeile 40. Da die Rufnummer vom Datentyp TCHAR ist, aber die Methode *sendData()* nur Attribute in Form von String's akzeptiert, erfolgt mit der Klasse *CT2CA* eine Datentypumwandlung (Zeile 43, 46).

```
53              case LINECALLSTATE_CONNECTED:
54                      //Verbindung aufgbaut
55                      m_CallInfoStrg.InsertString(-1,strZeit+(CString)" Verbindung wurde
hergestellt");
56                      //connect wird gesendet
57                      sendSocket->sendData(conn);
58                      break;
```

Nimmt der Anwender das Telefonat an und folgerichtig die Verbindung zwischen den zwei Teilnehmern herstellt ist, wird die Zustandskonstante LINECALLSTATE_CONNECTED in

dwParam1 abgebildet (Zeile 53). Dies führt zu einer Benachrichtigung *connect* an die Server-Anwendung (Zeile 57).

```
59              case LINECALLSTATE_IDLE:
60                      //Ruf existiert aber keine Verbindung zustande gekommen
61                      m_CallInfoStrg.InsertString(-1,strZeit+(CString)" Verbindung
abgebrochen");
62                      //onhook wird gesendet
63                      sendSocket->sendData(idle);
64                      //Socket schließen
65                      sendSocket->Close();
66                      break;
```

Für den Fall, dass der Anrufer frühzeitig die Verbindungsanfrage abbricht, meldet der Treiber mit LINECALLSTATE_IDLE den aktuellen Zustand (Zeile 59). Mit *onhook* wird die Nachricht stellvertretend für den Zustand an die Server-Anwendung geschickt (Zeile 63). Danach wird der Socket clientseitig geschlossen (Zeile 65).

```
67              case LINECALLSTATE_DISCONNECTED:
68                      //Verbindung beendet
69                      m_CallInfoStrg.InsertString(-1,strZeit+(CString)" Verbindung wurde
beendet");
70                      //disconnect wird gesendet
71                      sendSocket->sendData(disc);
72                      //Socket schließen
73                      sendSocket->Close();
74                      break;
75              }
76      }
77 }
```

Kam es zu einer Verbindung zwischen den Teilnehmern und wird diese beendet, wird der letzte Anweisungsblock ausgeführt (Zeile 67). Ein *disconnect* wird über den verbundenen Socket gesendet und dieser anschließend beendet (Zeile 71,73).

Anhand der Anweisungen in Zeile 20, 24, 28, 40, 49, 55, 61 und 69 erfolgt eine Anzeige in der grafischen Ausgabe der CTI-Anwendung bei Eintritt in den jeweiligen Zustand. Diese Funktionalität interagiert in keiner Weise mit den für das Systemkonzept erforderlichen Funktionen. Sie dienen lediglich der eventuellen Fehlersuche bei einer inkorrekten Codeausführung.

CTI-Steuerelemente für Rufnummer wählen und Hörer auflegen sind mit Ereignisfunktionen verknüpft und werden durch entsprechende Nachrichten von der Client-Anwendung im Media Center aufgerufen.

```
78 //CTI-Funktionalität für Rufnummer wählen
```

```
79 void CtapiPhoneDlg::OnBnClickedCall() {
80     UpdateData(true);
81     //Übergabe der benötigten Parameter an TAPI-Funktion
82     m_result = lineMakeCall(m_hLine,&m_hCall,m_PhoneNumberValue,0,NULL);
83
84     if(m_result<0){
85             m_CallInfoStrg.InsertString(-1,(CString)"Fehler beim Call");
86     }
87 }
88 //CTI-Funktionalität für Hörer auflegen
89 void CtapiPhoneDlg::OnBnClickedDrop() {
90     //Übergabe der benötigten Parameter an TAPI-Funktion
91     m_result = lineDrop(m_hCall,NULL,NULL);
92
93     if(m_result<0){
94             m_CallInfoStrg.InsertString(-1,(CString)"Error DropLine");
95     }
96 }
```

Nachdem die TAPI-Session initialisiert und das TAPI-Device für den entsprechenden Dienst (Sprache) geöffnet wurde, können die Funktionen zum Ausführen und Beenden eines Call's aufgerufen werden. Im Systemkonzept wird dies, wie schon erläutert, über eine empfangene Benachrichtigung durch den implementierten Server-Socket in TapiPhone realisiert. Die Ereignisfunktion *OnBnClickedCall()* (Zeile 79) enthält zum Ausführen eines Call's die TAPI-Funktion *lineMakeCall()* (Zeile 82). Parameter wie das Handle des geöffneten Line-Device *m_hline*, der Zielrufnummer *m_PhoneNumberValue* und der Länderkennung werden dabei der Funktion übergeben. Zurückgeliefert wird ein Zeiger auf das Handle des Call's *&m_hCall*. Gebraucht wird dies später zum Beenden des Call's. Empfängt der Server-Socket eine Benachrichtigung, den Call zu Beenden, wird die Ereignisfunktion *OnBnClickDrop()* aufgerufen (Zeile 89). Hierfür wird der TAPI-Funktion *lineDrop()* der Zeiger auf das Handle des Call's überreicht und mit Hilfe dessen der entsprechende Call beendet (Zeile 91).

5.1.2.2 MyAsyncSocket.cpp

Mittels der Implementierungsdatei *MyAsyncSocket.cpp* werden die Memberfunktionen der selbst definierten Klasse *CMyAsyncSocket* deklariert. Diese Klasse ist von der Originalklasse *CAsyncSocket* abgeleitet. Alle gesendeten oder empfangenden Daten des Client- beziehungsweise Server-Socket erfahren in dieser Datei eine weiterführende Bearbeitung.

```
1 // CMyAsyncSocket-Memberfunktionen
2 void CMyAsyncSocket::OnReceive(int nErrorCode) {
3     canReceive=true;
4     //Aufruf der Funktion zum Empfang der Daten und der weiterführenden Verarbeitung
```

```
5       receiveData();
6 }
```

Anhand der Memberfunktion *OnReceive()* gibt der Server-Socket bekannt, dass Daten mit dem Aufruf *Receive()* empfangsbereit sind (Zeile 2). Das bei der Initialisierung zu *false*-gesetzte Flag *canReceive* nimmt den Wert *true* an und die Funktion zum Empfang und Weiterverarbeitung der Daten wird aufgerufen (Zeile 3, 5).

```
7 void CMyAsyncSocket::OnSend(int nErrorCode){
8       canSend=true;
9 }
```

Mit *OnSend()* benachrichtigt der Client-Socket, dass unter Verwendung der Memberfunktion *Send()* Daten gesendet werden können. Im Körper der Funktion wird lediglich das Flag auf *true* gesetzt.

```
10 void CMyAsyncSocket::sendData(const char *text){
11      //Socket sendebereit?
12      if(canSend==true){
13              //Daten werden gesendet
14              Send(text,strlen(text));
15      }
16 }
```

Die in der TapiPhoneClg.cpp aufgerufene Memberfunktion *sendData()* zum Senden der Nachrichten (Zeile 46, 51, 57, 63, 71 in *TapiPhoneClg.cpp*) wird an dieser Stelle definiert (Zeile 10). Ist der Client-Socket zum Senden bereit, wird die Nachricht mit Hilfe der *Send()*-Funktion über den verbundenen Socket zur Server-Anwendung geschickt (Zeile 14). Mögliche Nachrichten sind, wie bereits erwähnt, *connect*, *disconnect*, *unknown* und *onhook* (Zeile 11 bis 14). Die Nachrichten präsentieren die Anrufzustände und werden durch entsprechende Inhalte in den Benachrichtigungsboxen in XBMC ausgegeben.

In Zeile 5 der Memberfunktion *OnReceive()* wird die Funktion *receiveData()* aufgerufen (Zeile 17).

```
17 void CmyAsyncSocket::receiveData(){
18      //Variablen anlegen
19      char buffer[1024];
20      int nRead;
21      CString message,number;
22      //Socket sendebereit?
23      if(canReceive==true){
```

Da der Server-Socket den Status empfangsbereit besitzt und demzufolge das Flag den Wert *true* angenommen hat, beginnt die Empfangsroutine mit der Memberfunktion *Receive()* (Zeile 25).

```
24              //Daten werden empfangen
25              nRead = Receive(&buffer,1024);
26              //Daten empfangen?
27              if(nRead==0) {
28                      Close();
29              }
30              else {
31                      //empfangene Daten aus Buffer lesen
32                      for(int i=0;i<nRead;i++){
33                              message+=buffer[i];
34                      }
```

Liegen Daten vor, wird der empfangene Bitstream Byte-weise ausgelesen und in das Objekt *message* geschrieben (Zeile 32, 33). Diese Vorgehensweise ist notwendig, um nur die reale Länge der empfangenen Nachricht aus dem 1024 Byte großen Puffer zu ermitteln und um später die Rufnummer aus dem Substring zu filtern. Ein Substring kann mittels der Klassenmethode *Find()* von *Cstring* ermittelt werden. Als Parameter übergibt man der Methode den zu durchsuchenden String und den Startindex dafür.

```
35                      //Nachricht dropcall empfangen?
36                      if (message.Find((CString)"dropcall")==0){
37                              //Ereignisfunktion aufrufen
38                              g_dlg->OnBnClickedDrop();
39                      }
40                      //Nachricht makecall empfangen?
41                      if (message.Find((CString)"makecall")==0){
42                              //Rufnummer aus String filtern
43                              number=message.Mid(9);
44                              //Rufnummer als Parameter der Ereignisfunktion übergeben
45                              g_dlg->m_PhoneNumberValue=number;
46                              g_dlg->UpdateData(false);
47                              //Ereignisfunktion aufrufen
48                              g_dlg->OnBnClickedCall();
49                      }
50                      else {
51                              Close();
52                      }
53              }
54      }
55 }
```

Enthält nun der empfangene String einen Substring namens *dropcall* (Zeile 36), so folgt der Aufruf der entsprechenden Ereignisfunktion (Zeile 38). Die Ereignisfunktion *OnBnClickedDrop()* wurde schon unter TapiPhoneClg.cpp in Zeile 89 erläutert. Lautet der Substring *makecall*, tritt die zweite Bedingung ein (Zeile 41). Hier muss nun die

Rufnummer aus dem Substring gefiltert werden. Realisiert wird dies über die Methode *Mid()* auf das Objekt *message* mit dem enthaltenen String *makecall:<rufnummer>* (Zeile 43). Der Startindex Neun zeigt auf die erste Zahl nach dem Doppelpunkt. Diese und jede weitere Zahl wird aus dem String gefiltert und in das Objekt *number* gespeichert. Mit dieser Rufnummer erfolgt der Aufruf der zweiten Ereignisfunktion (Zeile 48) unter TapiPhoneDlg.cpp in Zeile 79.

5.2 Installation und Einrichtung von freeVDR 3.0

Nachfolgend werden die Schritte von der Installation bis zur Konfiguration aller notwendigen Hard- und Softwarekomponenten der Distribution beschrieben.

5.2.1 Installation der Distribution

Nach dem Einlegen und Booten von der Installations-CD beginnt die Installationsroutine. Bei der Partitionsaufteilung sind einige Punkte zu beachten und werden auch von den Entwicklern empfohlen.

Insgesamt sind vier primäre Partition anzulegen. Partition Eins muss als Boot-Partition mit einer Größe von 64 MB gewählt werden. Die zweite Partition dient als Root-Partition und bekommt 8 GB zugeteilt. Partition Drei ist swap und wird mit 256 MB reserviert. Schließlich folgt die vierte Partition media mit der restlich verfügbaren Größe. Der folgende Überblick soll die Partitionsaufteilung auflisten:

- primaer boot 64 MB type 83 ext3
- primaer /root 8000 MB type 83 ext3
- primaer swap 256 MB type 82
- primaer /media REST type 83 ext3 [68]

Danach beginnt und läuft die Installation voll automatisch ab. Nach erfolgreicher Installation wird die CD ausgeworfen. Die Anmeldedaten lauten:

- Nutzer: root
- Passwort: freevdr

Schon nach wenigen Minuten steht dem Nutzer ein voll funktionsfähiges Media Center zur Verfügung. Es müssen lediglich noch Konfigurationen der Grafikkarte und zum DVB-T-Stick vorgenommen werden. Des Weiteren wird noch die Reihenfolge des Startvorgang's angepasst.

---

68 vgl. http://www.vdr-wiki.de/wiki/index.php/FreeVDR, Stand: 12.01.2011

5.2.2 Konfiguration von freeVDR

Ist der Installationsvorgang abgeschlossen, gelangt der Anwender auf die Xfce-Oberfläche.

Folglich sind passende Einstellungen für die Nutzung des VDR vorzunehmen. Zur Installation des nvidia-Treiber's ist eine Basis xorg.conf anzulegen. Dafür wird die Konsole geöffnet und folgende Befehle ausgeführt:

```
1 apt-get update
2 apt-get install nvidia-current
3 nvidia-xconfig
```

Damit ist die xorg.conf-Datei angelegt und der X-Server ausreichend konfiguriert. Weiterhin muss der DVB-T-Stick eingerichtet werden. Ob das Gerät ordnungsgemäß erkannt wurde und passende Treiber vorliegen, zeigt der folgende Befehl und ist in die Konsole einzutragen:

```
4 dmesg | grep DVB
```

Die Ausgabe zeigt eine erfolgreiche Initialisierung und Verbindung mit dem Gerät. Nun muss die Kanalliste erstellt werden. In der channels.conf-Datei sind Empfangs-Parameter zu verfügbaren Sendern in der Region hinterlegt. Der Befehl zum Scannen der Frequenzen und anschließendes Eintragen in die channels.conf-Datei lautet:

```
5 aptitude install dvb-apps
6 /etc/init.d/freevdr stop
7 scan -o vdr /usr/share/dvb/dvb-t/de-Sachsen > /var/lib/vdr/channels.conf
8 /etc/init.d/freevdr start
```

Das Plugin dvb-apps ist eine kleine Ansammlung von DVB-Programmen (Zeile 5). *scan* ist Bestandteil des Plugin's und führt einen manuellen Sendersuchlauf mit gleichzeitiger Erstellung der channels.conf aus (Zeile 7) [69]. Es ist zu beachten, dass vor dem Scannen keine Programme auf den DVB-Treiber zugreifen. Auch muss vorher der VDR gestoppt werden (Zeile 7). Eine fertige channels.conf ist im Anhang hinterlegt. Anhand dieser Einstellungen steht der Verwendung des VDR nichts mehr im Wege [70].

Zu guter Letzt ist die Sound-Hardware einzurichten. Die Distribution setzt in der Standardinstallation auf ALSA. Sound-Treiber werden von ALSA bereitgestellt und übernehmen so die Ansteuerung der Soundkarte. Jedoch muss ALSA im Systemkonzept um einen Soundserver erweitert werden. Ein Soundserver ist für die Weitergabe der Ton-

69 vgl. http://www.vdr-wiki.de/wiki/index.php/Dvb-apps, Stand: 12.01.2011

70 vgl. http://vdr-free.de/wiki/index.php?title=FreeVDR_3.0, Stand: 12.01.2011

Signale an die Treiber zuständig [71]. Zum Einsatz kommt hier PulseAudio. PulseAudio ist ein erweiterter Soundserver und kommt in vielen Ubuntu-Derivaten standardmäßig zum Einsatz. Der Soundserver kann über die Synaptic-Paketverwaltung nachinstalliert werden. Gibt der Anwender in der Schnellsuche *pulseaudio* ein, so werden alle benötigten Pakete bei der Auswahl automatisch nachgereicht. Allerdings besteht auch die Möglichkeit, das Paket über die Konsole zu installieren:

```
1 apt-get install pulseaudio
```

Zum Zeitpunkt der Installation von freeVDR 3.0 (Stand 19.8.10) war XBMC in der Version pvr2 -r 30311 enthalten. Diese Version enthielt keine Unterstützung des neuen Add-on-System's. Deshalb war das Entfernen dieser Version unabdingbar. Stattdessen viel die Entscheidung, wie schon im Systemkonzept erläutert, auf die Version mit integrierten PVR von Henning Pingel. Für diesen Schritt muss als Erstes das standardmäßig installierte XBMC mit Hilfe der Synaptic-Paketverwaltung deinstalliert und die Paketquelle aus den Software-Paketquellen von Drittanbietern entfernt werden. Zu finden sind die Punkte unter Start → System. Als Nächstes muss die Paketquelle von Henning Pingel unter Andere Software hinzugefügt werden:

```
1 deb http://ppa.launchpad.net/henningpingel/xbmc/ubuntu lucid main
2 deb-src http://ppa.launchpad.net/henningpingel/xbmc/ubuntu lucid main [72]
```

Nach Aktualisierung der Paketquelle ist die aktuelle Version von Henning Pingel in der Synaptic-Paketverwaltung zu finden und kann demzufolge installiert werden.

Zur anwendungsfreundlichen Konfiguration der Komponenten von freeVDR dient der Webkonfigurator, zu finden als Verknüpfung auf dem Desktop (Abbildung 5-12). Unter *Start* können Einstellungen zum Startverhalten von freeVDR und den VDR-Frontend-Programmen Xine und sxfe vorgenommen werden. Um dem Systemkonzept zu folgen, soll beim Bootvorgang von freeVDR das Media Center als Erstes starten. Demzufolge ist die Option anzuklicken. Die weiteren Einstellungen können so belassen werden. Eine Übersicht zu bereits installierten oder noch installierbaren Plugins für den VDR und XBMC bietet ebenfalls der Konfigurator. Über den Menüpunkt *Plugins* ist diese Übersicht aufrufbar. Ebenso können gleich aus der Übersicht weitere Plugins nachinstalliert oder aktualisiert werden. Unter *Logging* kann der Anwender in systembezogene Log-Dateien einschauen.

71 vgl. http://wiki.ubuntuusers.de/Soundsystem, Stand: 13.01.2011

72 vgl. https://launchpad.net/~henningpingel/+archive/xbmc, Stand: 05.01.2011

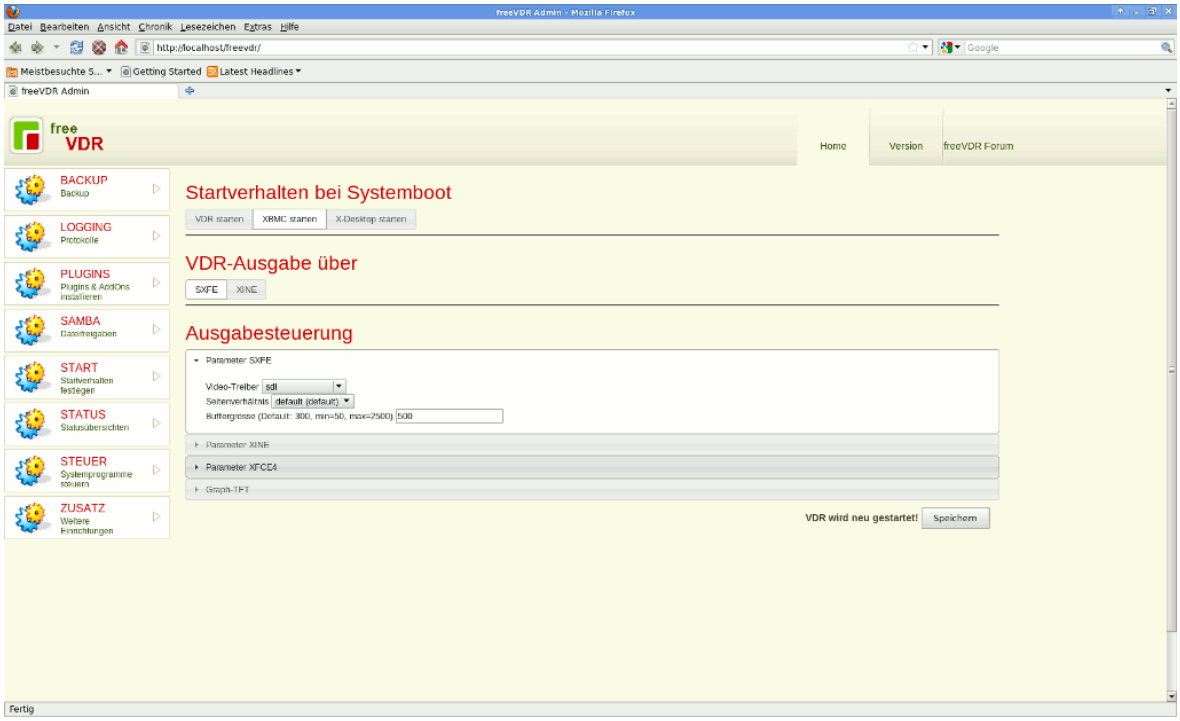

Abbildung 5-12: freeVDR Web-Konfigurator

5.2.3 Einrichten von XBMC

Der Startvorgang wurde nun so angepasst, dass als Erstes das Media Center aufgerufen wird. Auch hier sind noch Einstellungen zur Umsetzung der Anforderungen an das Systemkonzept vorzunehmen.

5.2.3.1 Sprache

Die Standardsprache Englisch kann natürlich in Deutsch verändert werden. Damit verbundene Einstellungen wie die Zeitzone werden in Folge automatisch angepasst.

- System → Darstellung → Sprache & Region

5.2.3.2 Audio

Für die Soundausgabe muss beachtet werden, dass die Ausgabe auf *analog* und das Ausgabegerät auf *default* gesetzt ist.

- System → System → Audio-Hardware

5.2.3.3 Live-TV und PVR-Client

Des Weiteren bedarf das VDR-Frontend unter XBMC noch einer Aktivierung. Nach der Aktivierung erscheint der Menüpunkt Live-TV im Homescreen.

- System → TV → Allgemein

Um den Dienst nutzen zu können, bedarf es allerdings noch einer Aktivierung des PVR-Client's. Hierfür bietet sich der VNSI-Server als Verbindung zwischen VDR und XBMC an. Der Client garantiert schnelle Umschaltzeiten und ist mit der XBMC-Version kompatibel.

- System → Add-ons → Aktivierte Add-ons → PVR clients

5.2.3.4 Wetter

Um auf regionale Wetterdaten zugreifen zu können, sind die Standardorte zu erweitern oder zu ersetzen. Zusätzlich kann der Wetterdienst, von dem XBMC die Wetterdaten bezieht, geändert werden.

- System → Wetter

5.2.3.5 Add-ons

Der Zugriff auf Online Bilder- und Video-Portale ist im Systemkonzept vorgesehen. Hierfür stehen dem Anwender bereits verschiedene Add-ons in Form von Plugins und Skripten bereit. Zwar muss der Anwender erst das Add-on aus dem Add-on-System installieren, doch dies geschieht aufgrund der Überarbeitung einfach und problemlos.

- System → Add-ons → Weitere Add-ons

Nach der Einrichtung kann das neue Add-on unter der entsprechenen Rubrik in *Bilder-Plugins* oder *Video-Plugins* ausgewählt werden.

5.3 Implementierung der Add-on's für XBMC

An dieser Stelle soll auszugsweise eine Erläuterung des implementierten Quellcodes sowie aller wichtigen Informationen bezüglich des Datenaustausch's zwischen der Client- und Server-Anwendung erfolgen.

Grundlagen in Python wurden im Kapitelabschnitt 3.2.3.1 betrachtet. Auf Basis dieser Einführung erfolgt für ausgewählte Quelltextabschnitte eine Erläuterung der Arbeitsweise dieser Auszüge. Zum Programmieren wurde ein einfacher Texteditor verwendet. Unter Ubuntu ist das gedit. Der vollständige Quelltext aller Implementierungen wird im Anhang hinterlegt.

5.3.1 Add-on zur Anrufsignalisierung und Anrufsteuerung

Zum besseren Verständnis zeigt Abbildung 5-13 die Softwarekomponenten, mit denen das Skript für bestimmte Ereignisse in Interaktion tritt. Die Verarbeitung wird nun im folgenden Kapitelabschnitt detailliert erläutert.

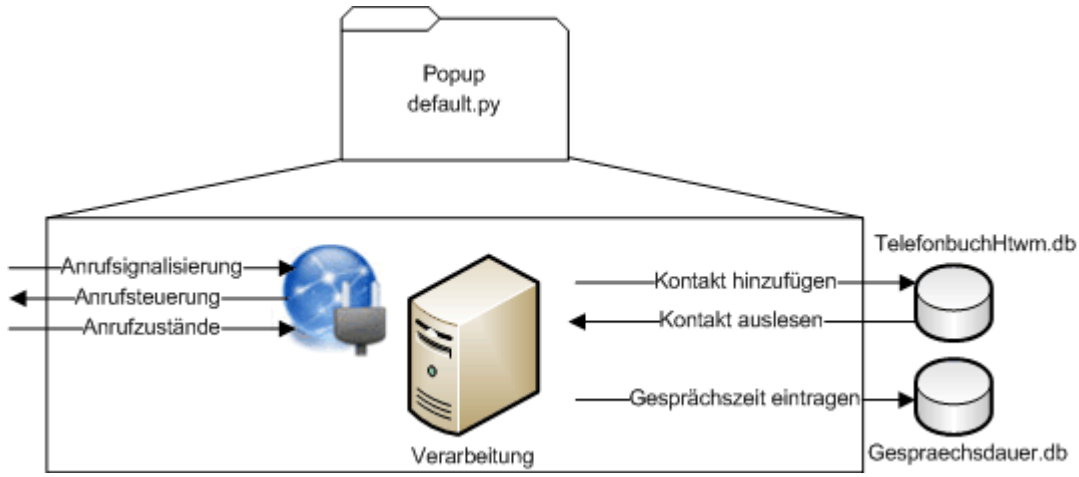

Abbildung 5-13: Interaktion des Add-ons mit den Softwarekomponenten

Um das Add-on beim Start von XBMC automatisch mitzustarten, bedarf es einer Zusatzdatei im Verzeichnis unter */root/.xbmc/userdata*. Diese Zusatzdatei, hier als Skript verwendet, wird beim Start durch den integrierten Python-Interpreter von XBMC aufgerufen. Unter dem Namen autoexec.py wird folgender Quelltext hinterlegt:

```
1 #Einbinden des Moduls
2 import xbmc
3 #Funktionsaufruf und Pfad als String übergeben
4 xbmc.executescript('/root/.xbmc/addons/script.popup/default.py')
```

Nach dem Einbinden des Modul's *xbmc* erfolgt der Aufruf des Add-on's *default.py* für die Anrufsignalisierung und Anrufsteuerung in Zeile 4. Der Pfad zum Add-on wird als Argument der Funktion übergeben. Durch das Abspeichern der Datei mit der Endung .py wird das Dokument als Python-Dokument deklariert.

Die Beschreibung des Add-on's durch die im Add-on-Ordner enthaltene XML-Datei entfällt. Grund ist hierfür der Hintergrundprozess.

Nun erfolgt die eigentliche Implementierung der Funktionalität des Add-on's in der Datei *default.py* im Ordner *script.popup* unter dem automatisch nach der Installation von XBMC angelegten addon-Ordner unter /root*/.xbmc/addons/script.popup*.

```
1 #Einbinden aller benötigten Module
2 import socket,asyncore,sqlite3,re,time
3 import xbmc,xbmcgui
```

Alle benötigten Module sind natürlich am Anfang einzubinden. Die Module *socket*, *asyncore* und *sqlite3* wurden schon hinreichend in vorherigen Abschnitten erläutert. Das Modul *re* der Standardbibliothek bietet Methoden und Funktionen für die Stringmanipulation an. In regulären Ausdrücken wird durch eine spezielle Syntax ein Textmuster beschrieben. Das Textmuster lässt sich dann auf verschiedene Textabschnitte

anwenden [73]. Durch das *time*-Modul können Funktionen für das Arbeiten mit Zeit- und Datumsangaben implementiert werden. Dies ist später für die Zeiterfassung des Gesprächszeitzähler's notwendig. In der 3. Zeile werden die Module der XBMC Python Engine eingebunden.

Im anschließenden Quelltextabschnitt wird der asynchrone Server-Socket eingerichtet.

```
4 #Klassendefinition
5 class Server-Socket(asyncore.dispatcher):
```

Anhand der Quelltextzeile 5 wird die Klassendefinition *Server-Socket* von der Objektoberklasse *asyncore.dispatcher* erzeugt.

```
6       #Aufruf der Konstruktormethode
7       def __init__(self,host,port):
8       ...
```

Die benötigten Socket-Dienste des Server-Socket's sind unter der Konstruktormethode ab Zeile 7 definiert. Als Argumente werden neben *self* auch noch *host* und *port* der Methode übergeben. Beim Instanziieren der Klasse *Server-Socket* am Ende des Quelltextes werden den beiden Argumenten die IP-Adresse und Port des lokalen Rechner's übergeben (Zeile 201).

```
9       #Methodendefinition zur Annahme der Clientverbindung
10      def handle_accept(self):
11      ...
```

Unter der *handle_accept()*-Methode werden mögliche Clientassoziationen behandelt und der Dispatcher gestartet (Zeile 13). Das Objekt *Client-Socket* beinhaltet als Argument die IP-Adresse des Clientrechner's.

```
12      #Dispatcher starten
13      SocketHandler(Client-Socket)
```

In Abhängigkeit der empfangenen Datensequenz aus dem errichteten Socket werden in der Klasse *SocketHandler* (Zeile 14) und innerhalb der *handle_read()*-Methode (Zeile 16) bestimmte Ereignisse ausgeführt.

```
14 class SocketHandler(asyncore.dispatcher):
15      #Methodendefinition zum Empfang und Verarbeitung der Nachrichten
16      def handle_read(self):
```

Mit dem in Zeile 17 gezeigten Auszug wird eine Instanz *dialog* der Klasse *Dialog* aus dem Modul *xbmcgui* erzeugt.

---

73 vgl. http://openbook.galileocomputing.de/python/python_kapitel_15_002.htm, Stand: 20.12.2010

```
17              dialog=xbmcgui.Dialog()
```

Auf die Instanz werden im späteren Verlauf Methoden zur Anzeige bestimmter Dialogboxen angewendet.

```
18              #Nachricht aus Socket empfangen
19              self.buffer=self.recv(1024)
```

Der Empfang der Nachricht vom Client-Socket geschieht mit der schon bekannten Methode *recv()* und ist auf 1024 Byte begrenzt (Zeile 19). Zwischengespeichert wird die Nachricht in *self.buffer*. Spätere Wahrheitsabfragen in Form von Bedingungen beziehen sich auf dieses Attribut.

Durch die folgenden *if*-Bedingungen erfolgt eine Selektion der empfangenen Nachrichten. So wird der Empfang der Nachricht *connect* in der ersten *if*-Anweisung ausgewertet (Zeile 21).

```
20              #Ruf angenommen = Hörer abgenommen
21              if self.buffer=="connect":
22                      #Built-In-Funktion für Benachrichtigungsbox
23                      xbmc.executebuiltin("Notification(Eingehender Anruf,Anruf
angenommen,5000,
 /root/.xbmc/userdata/notifications/test3.jpg)")
24                      #Prozessorzeit bei Annahme des Anrufs
25                      zeit1=time.clock()
```

Ist die Bedingung wahr, so erscheint nach der Anweisung in Zeile 21 eine Benachrichtigungsbox in XBMC (Zeile 23). Dieses Fenster mit Text, übergeben als Argument der Funktion, wird in Abbildung 5-14 gezeigt. Für das Bild muss der Pfad, unter dem das Bild lokal abgespeichert ist, übergeben werden. Im dritten Argument wird die Dauer der Anzeige festgelegt, hier mit 5 Sekunden.

Abbildung 5-14: Benachrichtigung bei Annahme

Außerdem wird die Startzeit des Anruf's für die Erfassung der Gesprächszeit benötigt. In Zeile 25 wird ein Objekt *zeit1* durch die Funktion *clock()* auf das *time*-Modul mit der Prozessorzeit erzeugt.

```
26              #Ruf beendet = Hörer aufgelegt
27              elif self.buffer=="disconnect":
28                      xbmc.executebuiltin("Notification(Eingehender Anruf,Anruf
beendet,5000,
/root/.xbmc/userdata/notifications/test2.jpg)")
```

```
29                      xbmc.executebuiltin("PlayerControl(Play)")
```

Sollte die Bedingung in Zeile 27 eintreffen, so beginnt die Abarbeitung mit der entsprechenden Benachrichtigungsbox unter Abbildung 5-15. Die Wiedergabe einer Datei wird wieder fortgesetzt (Zeile 29).

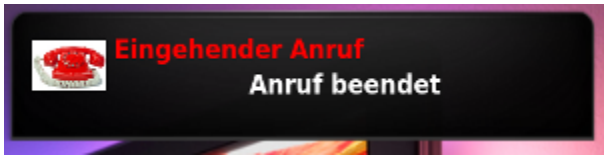

Abbildung 5-15: Benachrichtigung beim Auflegen

Danach folgt die Berechnung der Gesprächsdauer.

```
30                      if zeit1>0:
31                              #Prozessorzeit bei Beendigung des Gesprächs
32                              zeit2=time.clock()
33                              #Zeitdifferenz errechnen
34                              zeitdiff=zeit2-zeit1
```

Voraussetzung für die Abarbeitung dieser Schritte bildet eine Zeit größer 0 im *zeit1*-Objekt (Zeile 30). Zuerst wird die Prozessorzeit in ein zweites Objekt *zeit2* geschrieben (Zeile 32). Anschließend wird die Differenz der beiden Werte zum Anfang des Gespräch's und zum Ende des Gespräch's genommen (Zeile 34).

```
35                              #Zeitdifferenz kleiner 1 Minute?
36                              if zeitdiff/60<1:
37                                      #Dialogbox mit Anzeige der Gesprächszeit in Sekunden
aufrufen
38                                      ok=dialog.ok('Gesprächsdauer','Gesprächsdauer: %s
Sekunden' % int(zeitdiff))
```

Beträgt die Zeitdifferenz unter 60 Sekunden, erscheint ein Dialogfenster mit der Gesprächsdauer in Sekunden (Zeile 38). Wird die Methode *ok()* der Klasse *Dialog* auf die Instanz *dialog* dieser Klasse angewendet, wird ein Dialogfenster mit einen OK-Button generiert. Als Argumente werden der Methode Gesprächsdauer als Überschrift des Fenster's und der angezeigte Text übergeben. Da die Zeit als Gleitkommazahl ausgegeben wird, erfolgt noch eine Umwandlung in eine ganze Zahl *int*. Die nachfolgende Abbildung zeigt die Dialogbox.

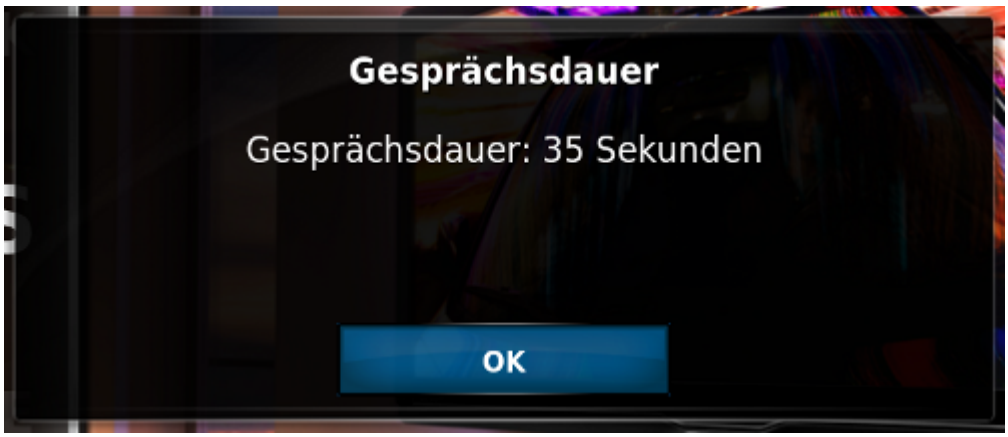

Abbildung 5-16: Dialogbox mit Anzeige der Gesprächszeit nach dem Anruf

```
39                          #Zeitdifferenz größer 1 Minute
40                          else:
41                                  #Berechnung der Minuten und Sekunden
42                                  zeitdiff_min=zeitdiff/60
43                                  zeitdiff_sec=zeitdiff%60
44                                  #Dialogbox mit Anzeige der Gesprächszeit in Minuten
und Sekunden
45                                  ok=dialog.ok('Gesprächsdauer','Gesprächsdauer: %s
Minuten und %s Sekunden' % (int(zeitdiff_min),int(zeitdiff_sec)))
```

Eine passende Anzeige in der Dialogbox wird auch bei einer Gesprächszeit über 60 Sekunden generiert. Zur Umrechnung in Minuten wird die Zeitdifferenz durch 60 dividiert und in ein neues Objekt *zeitdiff_min* geschrieben (Zeile 42). Für die restlichen Sekunden wird der Modulo-Operator verwendet (Zeile 43). Schließlich werden Minuten und Sekunden als Argumente der *ok()*-Methode übergeben.

```
46                          #Aufruf der Funktion zum Datenbankeintrag der Gesprächsdauer
47                          ct=calltimeSQLite(zeitdiff)
48                          #Rücksetzen der Zeit bei Annahme des Anrufs
49                          zeit1=0
```

Um die Gesprächsdauer später für den 3. Anwendungsfall im Systemkonzept, dem Gesprächszeit-Add-on, zur Verfügung zu stellen, bedarf es eines Datenbankeintrags. Die Funktion ist weiter unten im Quelltext implementiert und wird in Zeile 47 aufgerufen. Zu guter Letzt erfolgt ein Rücksetzen der Zeit bei Annahme des Telefonat's.

Beendet der rufende Teilnehmer vor der Annahme den Anruf, wird die nachfolgende Bedingung wahr.

```
50              #Anruf nicht angenommen
51              elif self.buffer=="onhook":
52                      xbmc.executebuiltin("XBMC.Notification(Eingehender Anruf,Anruf wurde
nicht angenommen,5000,/root/.xbmc/userdata/notifications/test2.jpg)")
```

```
53                      self.close()
```

Darauf hin wird dem Anwender der frühzeitige Abbruch der Verbindung signalisiert (Zeile 52), (Abbildung 5-17). Der Socket wird serverseitig geschlossen (Zeile 53).

Abbildung 5-17: Benachrichtigung bei Abbruch

In der nachfolgenden Bedingung wird ein eingehender Anruf ausgewertet. Der Anweisungsblock ist für den Fall der Rufnummernunterdrückung zuständig (Zeile 55).

```
54              #Rufnummer unterdrückt
55              elif self.buffer=="unknown":
56                      #Funktionsaufruf für aktuelle Zeit in Dialogbox
57                       cur_time=getTime()
58                      xbmc.executebuiltin("XBMC.Notification(Eingehender Anruf,Anruf von
unbekannt,10000,/root/.xbmc/userdata/notifications/unknown_contact.jpg)")
59                      #Built-In-Funktion zur Pausierung
60                      xbmc.executebuiltin("PlayerControl(Play)")
61                      #Auswahldialogbox generieren
62                      ret=dialog.yesno("Anruf Optionen",cur_time+": Anruf von
unbekannt","","Wollen Sie den Anruf unterdrücken?")
```

Zuerst wird das aktuelle Datum und die Uhrzeit durch die Funktion *getTime() (Zeile 197)* abgefragt und in *cur_time* geschrieben (Zeile 57). Eine Benachrichtigungsbox erscheint zur Information des Anrufer's mit dazugehörigen Text und Kontaktbild (Zeile 58).

Abbildung 5-18: Benachrichtigung über unbekannt

Sollte sich der Anwender gerade eine Sendung oder Film anschauen, pausiert diese Anwendung durch Aufruf der Built-In-Funktion für Pause (Zeile 60). Durch Aufruf der *yesno()*-Methode der Klasse *Dialog* wird eine Auswahlbox für zwei mögliche Optionen erzeugt (Zeile 62). Neben dem Text für Kopf und Rumpf der Dialogbox werden das zuvor ermittelte Datum und die Uhrzeit als Argumente übergeben. Der Anwender kann nun die weitere Vorgehensweise entscheiden. Entweder weist der Anwender den Anruf ab oder er nimmt den Anruf an. Folgende Abbildung zeigt die generierte Auswahlbox bei einen eingehenden Anruf.

Abbildung 5-19: Auswahlbox bei unterdrückter Rufnummer

```
63                      #Auswertung der Auswahl
64                      if ret==True:
65                              #Funktionsaufruf für Client-Socket
66                              getSocket('141.55.244.75',7777)
67                              xbmc.executebuiltin("Notification(Eingehender Anruf,Anruf
unterdrückt,5000,/root/.xbmc/userdata/notifications/test2.jpg)")
68                              xbmc.executebuiltin("PlayerControl(Play)")

69                      elif ret==False:
70                              pass
```

Mit dem Rückgabewert der Methode *yesno()* ist eine Bedingung verknüpft (Zeile 64). Durch Betätigung des Ja-Button's in der Dialogbox nimmt der Rückgabewert in *ret* den Wert *True* an. Demnach folgt der Funktionsaufruf *getSocket()*. Dort wird eine Assoziation zum Server-Socket auf Seiten von TapiPhone hergestellt und die entsprechende Nachricht gesendet. Daraufhin wird eine Benachrichtigungsbox mit entsprechenden Text und Bild erzeugt und die pausierte Anwendung fortgesetzt (Zeile 67, 68).

Abbildung 5-20: Benachrichtigung für unterdrückt

Andernfalls erfolgt bei Betätigung des Nein-Button's keine Abweisung. Die Anrufsignalisierung wird fortgesetzt. Beim Abnehmen des Hörer's erscheint die Benachrichtigungsbox unter Zeile 23.

Trifft keine der zuvor genannten Bedingungen ein, wird der folgende Anweisungsblock bei einem Anruf mit gesendeter Rufnummer im *else*-Zweig durchlaufen:

```
71              #Rufnummer mitgesendet
72              else:
```

```
73                  nmb=self.buffer
```

Der TAPI-Treiber übermittelt der CTI-Anwendung TapiPhone die Rufnummer des Anrufer's. Über den implementierten Client-Socket in TapiPhone gelangt die Rufnummer letztendlich in Form eines Strings an das Attribut *buffer* (Zeile 73).

```
74                  #Rufnummer auf x*0 prüfen
75                  if "0" in self.buffer[0:1]:
76                          #erste 0 wird abgeschnitten
77                          nmb=self.buffer[1:]
78                  else:
79                          pass
```

Beim Durchlaufen von diversen Tests bezüglich der korrekten Funktionalität der Anrufsignalisierung stellte sich ein Problem bei der Übermittlung der Rufnummer anhand des TAPI-Treiber's heraus. Erfolgt z.B. ein Anruf aus dem Fest- oder Mobilfunknetz heraus, so wird neben dem Präfix Null der Rufnummer und der Null aus Verkehrsausscheidungsziffer (Ferngespräche) zusätzlich eine Null angehängt. Wenn die Rufnummer nun in dieser Konstellation aus dem Telefonbuch heraus gewählt wird, kommt es zu keiner Verbindung mit dem Teilnehmer. Abhilfe schafft der Anweisungsblock in Zeile 75 und 77. Wird die Bedingung, an erster Stelle eine Null in der Rufnummer, wahr, wird diese Ziffer aus dem String abgeschnitten (Zeile 77).

```
80                  complcontact=getSQLite3Access(nmb)
```

Nun wird die Rufnummer mit den Telefonbucheinträgen in einer separat definierten Funktion verglichen (Zeile 80). Hierfür steht die Funktion *getSQLite3Access() (Zeile 108)* bereit. Die Instanz *complcontact* ist ein Listenobjekt. Der Funktion wird die in *nmb* gespeicherte Rufnummer übergeben.

```
81                  cur_time=getTime()
82                  #Ausgabe der Nachricht auf OSD und optionale Filmunterbrechung
83                  xbmc.executebuiltin("XBMC.Notification(Eingehender Anruf,Anruf von
%s,10000,%s)" % (complcontact[1],complcontact[2]))
```

Listenindex Eins von *complcontact* enthält je nach Vorhandensein des Kontaktes im Telefonbuch den Namen des Kontaktes oder die Rufnummer (Zeile 83). Listenindex Zwei von *complcontact* enthält den im Telefonbuch abgespeicherten Pfad zum Kontaktbild (Zeile 83) und die Benachrichtigungsbox wird mit dem Namen des Anrufer's und dem Kontaktbild zur Anzeige gebracht (Abbildung 5-21).

Abbildung 5-21: Benachrichtigung bei bekannt

Bei fehlenden Eintrag im Telefonbuch wird der Pfad für ein Standardbild verwendet und anstelle des Namen's die Rufnummer angezeigt (Abbildung 5-22) .

Abbildung 5-22: Benachrichtigung mit Rufnummer

```
84                      xbmc.executebuiltin("PlayerControl(Play)")
85                      ret=dialog.yesno("Anruf Optionen",cur_time+": Anruf von
"+complcontact[1],"Wollen Sie den Anruf unterdrücken?")
86                      if ret==True:
87                              getSocket('141.55.244.75',7777)
88                              xbmc.executebuiltin("Notification(Eingehender Anruf,Anruf
unterdrückt,5000,/root/.xbmc/userdata/notifications/test2.jpg)")
89                              xbmc.executebuiltin("PlayerControl(Play)")
90                      elif ret==False:
91                              pass
```

Abbildung 5-23: Auswahlbox bei bekannten Kontakt

Abbildung 5-23 zeigt die Auswahlbox nach dem erfolgreichen Abgleich der eingehenden Rufnummer mit dem Telefonbuchkontakt (Zeile 85).

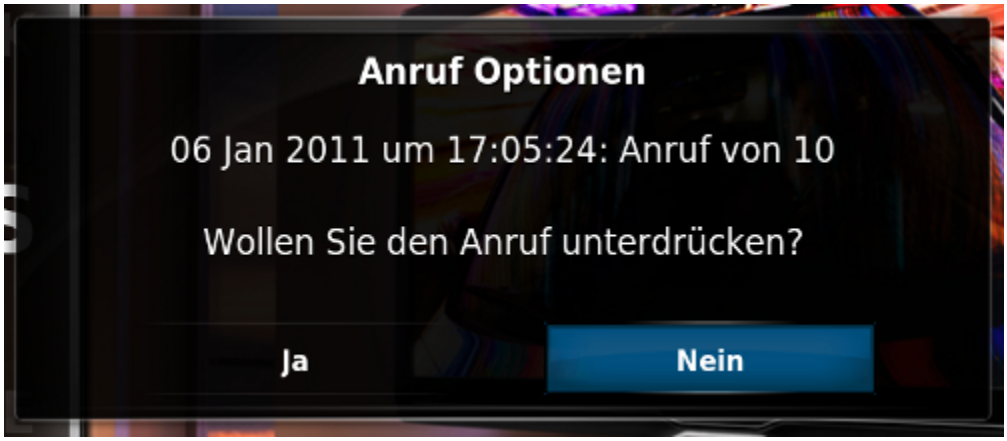

Abbildung 5-24: Auswahlbox bei unbekannten Kontakt

Entgegen der Abbildung 5-23 wird in Abbildung 5-24 die Rufnummer durch den fehlenden Eintrag im Telefonbuch dargestellt.

```
92                  #Kontakt ist nicht vorhanden
93                  if complcontact[0]==1:
94                          #Kontakt ins Telefonbuch eintragen?
95                          yesno=dialog.yesno("Telefonbuch","Kontakt in Telefonbuch
eintragen?")
96                          if yesno==True:
97                                  c=createContact(nmb)
98                          else:
99                                  pass
100                 #Eintrag vorhanden
101                 elif complcontact[0]==0:
102                         pass
```

Ist die Rufnummer mit verknüpften Kontakt nicht im Telefonbuch vorhanden, besteht die Möglichkeit eines Neueintrages durch die Funktion *createContact()* (Zeile 93, 144). In dieser Bedingung wird das Listenobjekt *complcontact* auf das Element mit Index Null auf den Wert Eins geprüft. Bei Wahrheit erfolgt nochmalig eine Abfrage an den Anwender durch eine Auswahlbox (Zeile 95).

Abbildung 5-25: Auswahlbox für Telefonbucheintrag

Bei Bestätigung wird die Funktion für den Eintrag aufgerufen (Zeile 97).

Im Anschluss werden die Funktionen zur Errichtung des Client-Socket's, zur Datenbankabfrage, zur Datum- und Zeitermittlung und zur Erstellung eines neuen Kontaktes im Telefonbuch definiert.

```
103 #Einrichten des Client-Socket für Anrufablehnung
104 def getSocket(host,port):
```

Anhand der Funktion *getSocket()* werden Methoden für die Initialisierung, zum Verbindungsaufbau und der Datenübertragung des Client-Socket's definiert. Die Vorgehensweise wurde schon im Kapitelabschnitt Fehler: Referenz nicht gefunden erläutert.

```
105     #Nachricht zur Ablehnung des Anrufs senden
106     csock.send("dropcall")
```

Es soll nur nochmal auf die zur CTI-Anwendung gesendete Nachricht eingegangen werden (Zeile 106). Die als String definierte Nachricht *dropcall* erfährt nach Empfang in TapiPhone eine spezielle Ereignisbehandlung und der Anrufer wird durch CTI-Funktionalität abgewiesen. Es folgt ein Besetztzeichen auf Seiten des Anrufer's.

```
107 #Datenbankabfrage für ankommenden Anruf
108 def getSQLite3Access(nmb):
```

Im weiteren Verlauf soll die Funktion zur Datenbankabfrage näher erläutert werden. Bleibt zu erwähnen, dass die Datenbank namens TelefonbuchHtwm.db schon im Vorfeld unter */root/.xbmc/userdata/Database/* angelegt wurde. Als Tool stand dem Entwickler dafür SQLiteman bereit.

```
109     #Anlegen einer leeren Liste
110     complcontact=[]
111     #Datenbankverbindung herstellen
112     connection=sqlite3.connect("/root/.xbmc/userdata/Database/
```

```
TelefonbuchHtwm.db")
113     #Cursor-Objekt erstellen
114     cursor=connection.cursor()
115     #gesendete Rufnummer mit möglicher Rufnummer in Datenbank vergleichen
116     cursor.execute("SELECT vorname,nachname,kontaktbild FROM telefonbuch WHERE
rufnummer=?",[nmb])
117     #Rückgabe des zutreffenden Datensatz
118     rows=cursor.fetchall()
```

Zu Beginn wird das Listenobjekt *complcontact* als leere Liste erzeugt (Zeile 110). Im weiteren Verlauf definiert die Funktion eine Methode zum Verbindungsaufbau mit der Datenbank (Zeile 112) und der Erstellung eines Cursor-Objektes für den Zugriff auf die abgelegten Datensätze (Zeile 114). Mittels der *execute()*-Methode des Cursor-Objektes wird das SQL-Statement *SELECT* zur Datenabfrage unter Angabe der Spalten *vorname*, *nachname*, *kontaktbild* und der Tabelle *telefonbuch* an die Datenbank gesendet (Zeile 116). Mit der *WHERE*-Klausel wird nur der Datensatz angefordert, auf den die gesendete Rufnummer mit der abgelegten Rufnummer in der Datenbank übereinstimmt. Anschließend wird der übereinstimmende Datensatz, bestehend aus Vorname, Nachname und Pfad zum Kontaktbild, ausgegeben (Zeile 118).

```
119     #Kontakt nicht vorhanden?
120     if not rows:
121             #Anweisung zur Übermittlung der Rufnummer
122             complcontact.append(1)
123             complcontact.append(nmb)
124             complcontact.append("/root/.xbmc/userdata/notifications/
unknown_contact.jpg")
```

Sollte der Datenbankabgleich keine Treffer bringen, wird der Anweisungsblock ab Zeile 122 abgearbeitet. Der leeren Liste wird auf Index Null eine Eins geschrieben (Zeile 122). Dieser Wert ist mit einen Flag vergleichbar. Ist der Wert auf Eins gesetzt, erfüllt es die Bedingung unter Zeile 93 und die Rufnummer kann durch einen neuen Telefonbucheintrag mit dem Kontakt verknüpft werden. Auf Listenindex Eins wird die Rufnummer für die spätere Anzeige in der Benachrichtigungsbox und der Auswahlbox geschrieben (Zeile 123, 83, 85), (Abbildung 5-21). Der Pfad zum Standardbild eines unbekannten Kontakt findet seinen Platz auf Listenindex Zwei und wird von dort aus abgerufen (Zeile 124, 83), (Abbildung 5-22).

```
125     else:
126             #Routine zur Übermittlung des Kontaktnamen aus Datenbank
127             complcontact.append(0)
128             for row in rows:
129                     contact=row[0]+' '+row[1]
130                     conpic=row[2]
131             #kein Kontaktbild vorhanden?
```

```
132             if not row[2].endswith('.jpg') or row[2].endswith('.png') or
row[2].endswith('.gif'):
133                     conpic="/root/.xbmc/userdata/notifications/
empty_contact.jpg"
134             else:
135                     pass
136     #Listenobjekt complcontact generieren
137     complcontact.append(contact)
138     complcontact.append(conpic)
```

Befindet sich die Rufnummer im Telefonbuch, wird der Anweisungsblock übersprungen und die Routine ab Zeile 127 durchlaufen. Im Gegensatz eines nicht vorhandenen Kontakt beträgt der Wert auf Listenindex Null nun Null (Zeile 127). Dies hat allerdings keine Auswirkung auf weitere Ereignisse (Zeile 101). Nachfolgend wird das Ergebnis-Tupel durch iterieren aus dem Listenobjekt *rows* angefordert (Zeile 128). Das Tupel besteht aus dem Vorname auf Index Null, dem Nachname auf Index Eins und dem Pfad auf Index Zwei. Der Vorname und Nachname wird mit Leerzeichen dem Objekt *contact* zugeordnet (Zeile 129). Ebenfalls wird der Pfad dem Objekt *conpic* zugewiesen (Zeile 130). Sollte kein Kontaktbild zum Kontakt hinterlegt sein, wird anstelle dessen ein Standardbild übergeben (Zeile 132). Überprüft wird dies auf die Endung des String's. Endet der Pfad nicht mit den Dateiendungen .jpg, .png oder .gif, so erfolgt eine Zuweisung des Standardbildes an das Objekt (Zeile 133). Abschließend wird dem Listenobjekt *complcontact* der Vorname und Nachname auf Index Eins und der Pfad auf Index Zwei geschrieben (Zeile 137, 138).

```
139     #Datenbankverbindung beenden und Listenobjekt zurückgeben
140     cursor.close()
141     connection.close()
142     return complcontact
```

Nach der Routine wird das Cursor-Objekt geschlossen und die Verbindung zur Datenbank beendet. Das Listenobjekt mit Vorname, Nachname und Kontaktbild beziehungsweise der Rufnummer und Kontaktbild wird zurückgegeben (Zeile 142, 80).

Zum Hinzufügen von Kontakten dient die Funktion *createContact()* und wird in den folgenden Zeilen erläutert.

```
143 #Kontakt in Telefonbuch anlegen
144 def createContact(nmb):
145     dialog=xbmcgui.Dialog()
146     connection=sqlite3.connect("/root/.xbmc/userdata/Database/
TelefonbuchHtwm.db")
147     cursor=connection.cursor()
```

Natürlich muss neben der Instanziierung der Klasse *Dialog* für das Generieren einer Dialog- oder Auswahlbox eine Verbindung zur Datenbank und die Erstellung eines Cursor-Objektes erfolgen (Zeile 144 bis 147).

```
148   #virtuelle Tastatur erzeugen
149   kb=xbmc.Keyboard(nmb,'Rufnummer übernehmen?',False)
150   # virtuelle Tastatur auf OSD anzeigen
151   kb.doModal()
152   if (kb.isConfirmed()):
153           #Rufnummer an Objekt übergeben
154           ruf=kb.getText()
155   kb=xbmc.Keyboard('','Bitte geben Sie den Vornamen ein',False)
156   kb.doModal()
157   if (kb.isConfirmed()):
158           #Vorname an Objekt übergeben
159           vor=kb.getText()
160   …
```

Virtuelle Tastaturen ermöglichen dem Anwender, benutzerfreundlich Einträge hinzuzufügen. Zuerst muss eine Instanz der *Keyboard*-Klasse erzeugt werden (Zeile 149). Daneben wird schon die Rufnummer als Default-Text angezeigt. Zuständig dafür ist das erste Argument in der Argumentliste. Zweites Argument präsentiert die Kopfzeile der Tastatur. Mittels der Methode *doModal()* wird die Tastatur zur Anzeige gebracht. Bricht der Anwender die Eingabe nicht ab, wird die Rufnummer an das Objekt *ruf* übergeben (Zeile 159). Selbige Routine läuft für den Eintrag des Vornamen und Nachnamen ab.

```
161   #Browser-Dialog für Pfad zum Kontaktbild an Objekt übergeben
162   picfile=dialog.browse(1,'Bitte Pfad zum Kontaktbild auswählen','myprograms','.jpg|.png',
True,False,'/root/.xbmc/userdata/notifications/Kontaktbilder')
```

Zur Bekanntmachung des Pfades zum Kontaktbild erscheint ein Browser-Dialog. Der Pfad wird ebenfalls in ein Objekt gespeichert (Zeile 162) (Abbildung 5-26).

Abbildung 5-26: Browser-Dialog für Kontaktbild

```
163    #String für Datenbankeintrag bearbeiten
164    text=(vor+' '+nach+' '+ruf+' '+picfile)
165    liste=re.split(r"\s",text)
```

Um für den parametrisierten Datenbankeintrag ein Tupel der *execute*-Methode übergeben zu können, werden die Einträge als String, getrennt durch Leerzeichen, in das *text*-Objekt geschrieben. Darauf folgend wird der String nach Übereinstimmungen mit dem regulären Ausdruck *r"\s"* durchsucht (Zeile 164). Passende Teilstring's werden als Trennzeichen angesehen. Dazwischenliegende Teile werden als Liste von Teilstring's in das Objekt *liste* zurückgegeben (Zeile 165).

```
166    #Datenbankeintrag
167    sql="INSERT INTO telefonbuch VALUES (?,?,?,?)"
168    cursor.execute(sql,liste)
169    yesno=dialog.yesno('Telefonbucheintrag','Kontakt wirklich zum Telefonbuch
hinzufügen?')
170    if yesno==True:
171            #Eintrag bestätigen
172            connection.commit()
173            xbmc.executebuiltin('Notification(Telefonbuch,Kontakt wurde hinzugefügt,5000,
/root/.xbmc/userdata/notifications/db_add.png)')
174            cursor.close()
175            connection.close()
176    else:
177            cursor.close()
178            connection.close()
```

Für den Datenbankeintrag werden im Query-String statt den Parametern an den ausgewählten Stellen Fragezeichen gesetzt (Zeile 167) und als ersten Parameter der *execute()*-Methode übergeben (Zeile 168). Dann übergibt man noch der *execute()*-Methode das Tupel in *liste* und die Werte werden vorübergehend in den Arbeitsspeicher übernommen (Zeile 168). Schließlich muss der Eintrag anhand der *commit()*-Methode des *connection*-Objekt's endgültig in die Datenbank eingetragen werden. Über eine Auswahlbox und der Bestätigung mit *Ja* wird dies vollzogen (Zeile 169). Ein Benachrichtigungsfenter informiert den Anwender über den ausgeführten Eintrag (Zeile 173) (Abbildung 5-27). Anschließend wird die Verbindung zur Datenbank getrennt.

Abbildung 5-27: Benachrichtigung für Eintrag

Die Funktion zum Eintragen der Gesprächsdauer in die Datenbank wird bei Empfang von *connect* aufgerufen, *zeitdiff* mit der beinhalteten Zeitdifferenz übergeben (Zeile 47) und hier abgearbeitet. Ebenfalls wurde die SQLite3-Datenbank namens Gespraechsdauer.db unter bekannten Pfad schon im Vorfeld angelegt. Sie besteht aus der Tabelle *gespraechszeit* und der Spalte *sekunden* und *minuten*.

```
179 #Gesprächsdauer in Datenbank eintragen
180 def calltimeSQLite(zeitdiff):
181     connection=sqlite3.connect("/root/.xbmc/userdata/Database/
Gespraechsdauer.db")
182     cursor=connection.cursor()
```

Wie schon in mehrfach erwähnt, muss am Anfang die Verbindung zur Datenbank und das Cursor-Objekt hergestellt beziehungsweise generiert werden (Zeile 180 bis 182).

```
183     #Zeit unter 60 Sekunden
184     if zeitdiff/60<1:
185             time=(int(zeitdiff),0)
186             cursor.execute("INSERT INTO gespraechszeit VALUES (?,?)",time)
187     #Zeit über 60 Sekunden
188     else:
189             zeitdiff_min=zeitdiff/60
190             zeitdiff_sec=zeitdiff%60
191             time=(int(zeitdiff_sec),int(zeitdiff_min))
192     cursor.execute("INSERT INTO gespraechszeit VALUES (?,?)",time)
193     connection.commit()
194     cursor.close()
195     connection.close()
```

Die Bedingung unter Zeile 184 wird wahr, wenn die Gesprächszeit unter 60 Sekunden beträgt. Auch hier wird der *execute()*-Methode im Query-String anstelle der Parameter Fragezeichen als Platzhalter übergeben, um mit Hilfe des *time*-Objekt's die ganzzahlige Zeit in Sekunden einzutragen (Zeile 185, 186). In der Spalte *minuten* wird folgerichtig eine Null eingetragen. Bei einer Gesprächsdauer über 60 Sekunden folgt der Anweisungszweig ab Zeile 189. Die in Sekunden betragende Zeitdifferenz wird für den Erhalt der Minuten durch 60 dividiert und in *zeitdiff_min* geschrieben (Zeile 189). Restliche Sekunden ermittelt der Modulo-Operator und schreibt den Wert in *zeitdiff_se*c (Zeile 190). Die Routine zum Datenbankeintrag ist mit der im ersten Anweisungsblock identisch. Nun wird der Eintrag noch durch die *commit()*-Methode übernommen und die Methoden zur Beendigung der Verbindung folgen anschließend.

```
196 #Funktion zur aktuellen Zeitermittlung
197 def getTime():
198     cur_time=strftime("%d %b %Y um %H:%M:%S",gmtime())
199     return cur_time
```

Für die Zeitangabe des ankommenden Anruf's über die Auswahlbox (Zeile 62, 85) steht die Funktion ab Zeile 197 bereit. Dafür wandelt die Funktion *strftime()* die *time.struct_time*-Instanz mit koordinierter Weltuhr in einen String um (Zeile 198). Platzhalter im Format-String werden im Ergebnis durch die entsprechenden Werte ersetzt.

```
200 #Generierung des Sockets
201 s=ServerSocket('141.55.243.248',6666)
202 #Socket Polling
203 asyncore.loop(1)
```

Nun wird der Server-Socket instanziiert (Zeile 201) und demzufolge gestartet. Zu guter Letzt erfolgt der Schleifenaufruf. Die Anzahl der Durchgänge, bis die Schleife beendet oder alle geöffneten Kanäle geschlossen werden, wird auf Eins festgelegt (Zeile 203).

5.3.2 Add-on für das Telefonbuch

Auch hier soll eine einleitende Abbildung für einen besseren Überblick der Softwarekomponenten und deren verknüpften Ereignissen dienen. Die Umsetzung der Verarbeitung wird im folgenden Kapitelabschnitt realisiert.

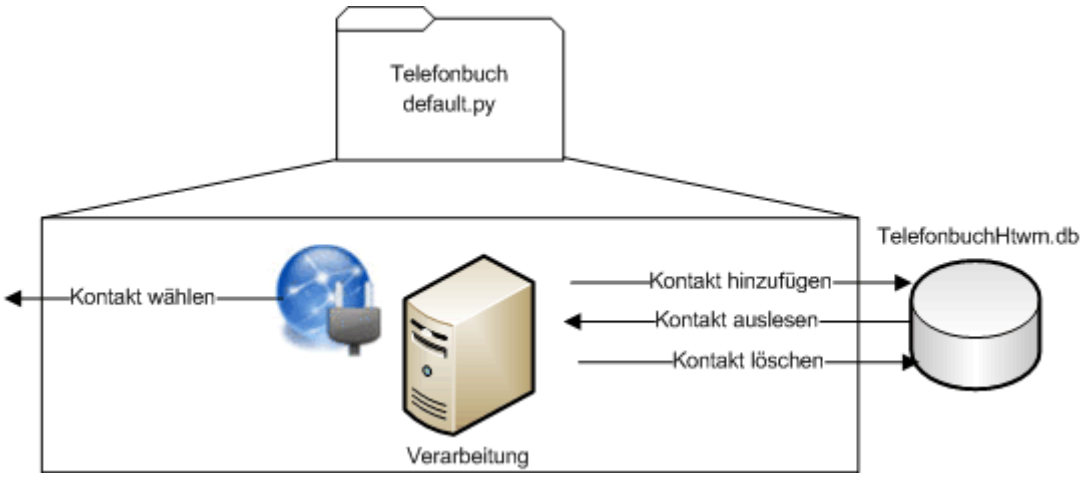

Abbildung 5-28: Interaktion des Add-ons mit den Softwarekomponenten

Wie bereits im Kapitelabschnitt 4.5.2.1.1 erwähnt, muss das Add-on aufgrund des neuen Add-on-System's unter einer spezifischen Ordnerstruktur abgespeichert und in einer XML-Datei beschrieben werden. Das Add-on befindet sich unter den Pfad: */root/.xbmc/addons/script.program.telephonebook*. Die im Add-on-Ordner hinterlegte XML-Datei *addon.xml* beschreibt das Add-on und hat folgende Struktur und Inhalt:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <addon
3       id="script.program.telephonebook"
4       name="Telefonbuch"
5       version="1.0.0"
6       provider-name="Matthias Baier">
7   <requires>
8     <import addon="xbmc.python" version="1.0"/>
9   </requires>
10   <extension point="xbmc.python.script"
11           library="default.py">
12   </extension>
13   <extension point="xbmc.addon.metadata">
14     <platform>all</platform>
15     <summary lang="en">Plugin phone book</summary>
16     <description lang="en">Plugin provides a phone book with call option</description>
17   </extension>
18 </addon>
```

Nach der typischen Kopfzeile in XML folgt das *addon*-Element und listet vier Attribute nacheinander auf (Zeile 2). Das *id*-Attribut ist für die einzigartige Kennung des Add-on's verantwortlich und repräsentiert meist den Namen des Add-on-Ordner's (Zeile 3). Sonderzeichen sind dabei nicht erlaubt. Mit dem zweiten Attribut *name* definert der Entwickler den in XBMC angezeigten Namen. Für das spezifische Add-on wird der Name Telefonbuch vergeben (Zeile 4). Unter *version* steht die aktuelle Version 1.0.0 des Add-

on's und wird von XBMC verwendet, um festzustellen, ob Updates vorliegen (Zeile 5). Der Entwickler des Add-on's kann sich unter *provider-name* eintragen (Zeile 6). Als Grundlage dieses Add-on dient das *xbmc-Python*-Add-on in seiner Version 1.0 und wird in Zeile 8 unter *requires* verankert. Mit *xbmc.python.script* ist das Standard-Erweiterungs-Skript beschrieben (Zeile 10). Damit wird der Teil, um das sich XBMC durch das Add-on erweitert, beschrieben. Unter *library* ist der Dateiname des Add-on's angegeben (Zeile 11). Um Anwender Zusatzinformationen bereitzustellen, dient der Erweiterungspunkt *xbmc.addon.metadata* (Zeile 13). Hier werden Informationen zur Computerplattform, auf dessen das Add-on läuft (Zeile 14) und kurze, zusammenfassende Beschreibungen zum Add-on gegeben (Zeile 15,16).

Der eigentliche Quelltext, mit dem das Add-on umgesetzt wird, ist unter dem Dateinamen *default.py* im Add-on-Ordner hinterlegt und wird nachfolgend erläutert. Da sich einige Auszüge mit denen vorherig erläuterten ähneln, soll nur auf neue eingegangen werden. Der vollständige Quelltext wird im Anhang hinterlegt.

```
1 #Definition des Client-Socket
2 class Client-Socket(object):
3       def __init__(self,host,port,callid):
4               csock=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
5               csock.connect((host,port))
6               #Nachricht für Anruf tätigen mit Angabe der Rufnummer
7               csock.send("makecall:"+callid)
8               csock.close()
```

Nach dem obligatorischen Einbinden der benötigten Module folgt die Definition des Client-Socket's (Zeile 2) für den Anruf. Nach der üblichen Prozedur mit Initialisierung und Verbindungsaufbau zum Server-Socket wird der Kontakt unter Angabe der Rufnummer mit Hilfe der TAPI-Funktion in TapiPhone gewählt. Diese Option wird mit der Methode *send()* und den Argumenten *makecall* und *callid* realisiert (Zeile 7). Die Rufnummer ist in *callid* gespeichert. Die Klassendefinition musste durch eine Fehlermeldung beim Interpretieren des Quelltextes an erster Stelle platziert werden. Ansonsten konnte bei Aufruf der Klassedefinition *Client-Socket* in der Programmablaufroutine diese nicht gefunden werden.

```
9 cursor.execute("SELECT vorname,nachname,rufnummer,kontaktbild FROM telefonbuch")
10 rows=cursor.fetchall()
11 #Auswahlliste um Kontaktoptionen hinzufügen und löschen erweitern
12 contacts=['Kontakt hinzufügen','Kontakt löschen']
13 #leere Liste numbers und conpics anlegen
14 numbers=[]
15 conpics=[]
16 for row in rows:
17       contact=row[0]+' '+row[1]
18       number=row[2]
```

```
19      conpic=row[3]
20      contacts.append(contact)
21      numbers.append(number)
22      conpics.append(conpic)
23 #Kontaktoptionen und Kontakte anzeigen
24 ret=dialog.select("Bitte Kontakt auswählen",contacts)
```

Bei Aufruf des Add-on's findet ein Aufbau der Verbindung zur Datenbank *Telefonbuch.db* und das Abrufen vorliegender Kontakte mit Vorname, Nachname, Rufnummer und Pfad zum Kontaktbild statt (Zeile 9). Mögliche Einträge werden nach den Kontaktoptionen Löschen und Hinzufügen der Liste angehängt und zur Anzeige gebracht (Zeile 23). Dafür wird dem Listenelement *contacts* auf Index Null und Eins der Eintrag Kontakt hinzufügen beziehungsweise Kontakt löschen angehängt (Zeile 12). Nach der Routine für das Tupelweise abspeichern der Datensätze (Zeile 16) werden die Kontakte der Auswahlliste angehängt. Unter Abbildung 5-29 ist die Auswahlliste dargestellt. Mittels der *select()*-Methode auf die Instanz der Klasse *Dialog* lässt sich solch eine Liste, unter Angabe der Argumente für die Überschrift und den Listenelementen, generieren (Zeile 23).

Abbildung 5-29: Auswahlliste im Telefonbuch

```
25 #Auswahl Kontakt hinzufügen
26 if ret==0:
```

```
27      kb=xbmc.Keyboard('','Bitte geben Sie den Vornamen ein',False)
28      kb.doModal()
29      if (kb.isConfirmed()):
30              vor=kb.getText()
```

Die Methode *select()* liefert Rückgabewerte der einzelnen Zeilen. So besteht die Zeile Eins aus dem Rückgabewert Null, Zeile Zwei aus Eins usw. Klickt der Anwender auf das Element Kontakt hinzufügen, springt der Interpreter demzufolge in die Anweisung auf Zeile 26. Hier erfolgen ebenfalls die Anweisungsschritte für Nachname, Rufnummer und Pfad zum Kontaktbild wie unter Zeile 27 bis 30 für den Vorname. Lediglich ändert sich das Objekt, in das der eingegebene Text gespeichert wird (Zeile 30). Zum Schluss der Anweisungsfolge wird der Eintrag nach den bekannten Methodenaufrufen der Datenbank hinzugefügt und die Verbindung beendet.

```
31 #Auswahl Kontakt anrufen
32 if ret>=2:
33      callid=numbers[ret-2]
34      callcon=contacts[ret]
35      callpic=conpics[ret-2]
36      if not callpic.endswith('.jpg') or callpic.endswith('.png') or callpic.endswith('.gif'):
37              callpic="/root/.xbmc/userdata/notifications/empty_contact.jpg"
38      else:
39              pass
40      #Rufnummer wählen
41      c=Client-Socket('141.55.244.75',7777,callid)
42      xbmc.executebuiltin('Notification(Ausgehender Anruf,%s wird angerufen,10000,%s)' %
(callcon,callpic))
```

Klickt der Anwender auf einen Kontakt, so wird dieser mit verknüpfter Rufnummer gewählt. Alle verfügbaren Kontakte mit dazugehörigen Rückgabewert größer Zwei werden in dieser Anweisung verarbeitet (Zeile 32). Grund hierfür sind die zuvor erläuterten Elemente (Zeile 12). Die Rufnummer wird anhand des Rückgabewertes des entsprechenden Kontakt minus Zwei ermittelt (Zeile 32). Wird z.B. der Kontakt an dritter Stelle der Auswahlliste mit Rückgabewert Zwei gewählt, ergibt sich die dazugehörige Rufnummer aus Listenindex Null des Listenobjektes *numbers* (Rückgabewert minus Zwei). Also genau die Nummer, die in Zeile 20 auf dem Listenindex Null der leeren Liste geschrieben wurde. In Folge wird die ermittelte Rufnummer der Klassendefinition *Client-Socket* übergeben (Zeile 42).

Als letzte Option steht dem Anwender gemäß Systemkonzept das Löschen von Kontakteinträgen aus dem Telefonbuch zur Verfügung.

```
43 #Auswahl Kontakt löschen
44 if ret==1:
45      cursor.execute("SELECT vorname,nachname FROM telefonbuch")
```

```
46    rows=cursor.fetchall()
47    contacts=[]
48    for row in rows:
49            contact=row[0]+' '+row[1]
50            contacts.append(contact)
51    #neue Auswahlliste generieren
52    ret=dialog.select("Kontakt löschen",contacts)
53    #Auswahl des zu löschenden Kontakts anhand der Rufnummer
54    removeid=numbers[ret]
55    cursor.execute("DELETE FROM telefonbuch WHERE rufnummer=?",[removeid])
56    yesno=dialog.yesno('Telefonbucheintrag','Kontakt aus dem Telefonbuch löschen?')
```

Hierbei wird eine neue Auswahlliste mit den verfügbaren Kontakten generiert und zur Anzeige gebracht (Zeile 46 bis 53). Nun wird mit Hilfe der Rufnummer der zum Löschen gewünschte Kontakt eindeutig beschrieben (Zeile 55). Dies erfolgt wieder durch den Rückgabewert des entsprechenden Kontakt. Nur wird diesmal der Index des Listenelements *numbers* im Original übernommen, da die vordefinierten Elemente fehlen. Mit dem SQL-Befehl *DELETE* wird der Eintrag gelöscht, in dessen Spalte *rufnummer* die Rufnummer des zu löschenden Kontakt steht. Nach Bestätigung des Eintrag's mit der Auswahlbox erscheint die Benachrichtigung unter Abbildung 5-30.

Abbildung 5-30: Benachrichtigung für Eintrag gelöscht

5.3.3 Add-on für die Gesprächszeit

Abgesehen von der Anrufsignalisierung, Anrufsteuerung und dem Telefonbuch wird eine weitere Anforderung an das Systemkonzept gestellt, dem Gesprächszeitzähler. In den folgenden Zeilen wird die Umsetzung erläutert. Der vollständige Quelltext ist wieder im Anhang zu finden. Die Datenbank mit der Tabelle *gespraechszeit* und den Spalten s*ekunden*, *minuten* wurde im Vorfeld angelegt. Wie schon in den beiden anderen Anwendungsfällen aufgezeigt, soll wieder die Abbildung 5-31 für einen besseren Überblick dienen. Erläutert wird die Verarbeitung in den folgenden Zeilen.

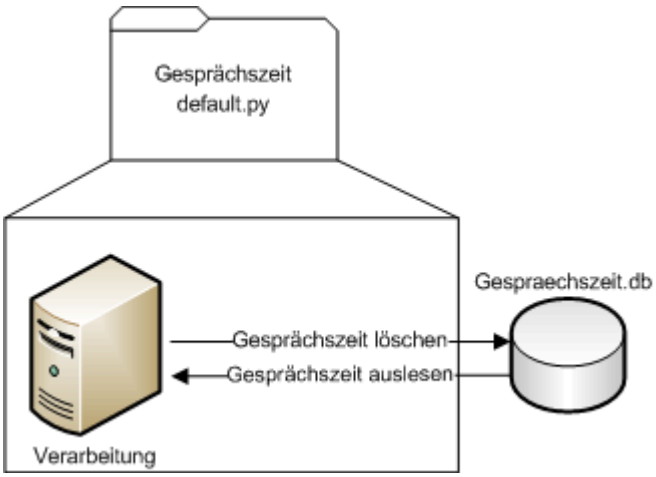

Abbildung 5-31: Interaktion des Add-ons mit den Softwarekomponenten

Wie schon das Telefonbuch-Add-on muss auch dieses in einem eigenen Ordner abgespeichert und mit der XML-Datei beschreiben werden. Zu finden ist das Add-on unter */root/.xbmc/addons/script.program.calltime* und die darin enthaltene XML-Datei unterscheidet sich unwesentlich von der des Telefonbuch's. Lediglich muss das *id*-Attribut mit dem Ordner-Namen und das *name*-Attribut ersetzt werden. Die Zusatzinformation sind ebenfalls anzupassen. Im Anhang wird die XML-Datei aufgezeigt.

```
1 #Listenelement definieren
2 totaltime=["Gesprächszeit zurücksetzen"]
```

Am Anfang wird das später zur Darstellung verwendete Listenelement *totaltime* um die Rücksetzfunktion erweitert (Zeile 2).

```
3 #Werte aus Datenbank holen, summieren und in Sekunden umrechnen
4 cursor.execute("SELECT sekunden,minuten FROM gespraechszeit")
5 rows=cursor.fetchall()
6 for row in rows:
7       sec=row[0]
8       minu=row[1]
9       calltime_sec.append(sec)
10      calltime_min.append(minu)
11 addtime_sec=sum(calltime_sec)
12 addtime_min=sum(calltime_min)
13 addtime_msec=addtime_min*60
14 min_sec=addtime_msec+addtime_sec
```

Nachdem die Verbindung zur Datenbank hergestellt und benötigte Objekte initialisiert wurden, folgt die Summierung der Werte. In das Objekt *addtime_sec* wird die Summe aus allen Einträgen der Datenbankspalte *sekunden* geschrieben (Zeile 9). Gleiches erfolgt mit der Datenbankspalte *minuten* und dem Objekt *addtime_min* (Zeile 12). Nur wird danach der Wert zur besseren Verarbeitung mit 60 multipliziert und somit in Sekunden

umgerechnet (Zeile 13). Zum Schluss werden die zwei Werte wieder summiert und in *min_sec* abgelegt.

```
15 #Verarbeitung der Sekunden
16 if min_sec/60<1:
17     total_temp_s=str(min_sec)+' Sekunden '
18     totaltime.append(total_temp_s)
19 if min_sec/60>1 and min_sec/3600<1:
20     total_min=min_sec/60
21     total_sec=min_sec%60
22     total_temp_m=str(total_min)+' Minuten : '+str(total_sec)+' Sekunden '
23     totaltime.append(total_temp_m)
24 else:
25     total_h=min_sec/3600
26     total_min=total_h/60
27     total_sec=total_h%60
28     total_temp_h=str(total_h)+' Stunden :'+str(total_min)+' Minuten : '+str(total_sec)+'
Sekunden '
29     totaltime.append(total_temp_h)
30 #Ausgabe
31 ret=dialog.select("Gesprächszeit",totaltime)
```

Ab Zeile 16 beginnt die Verarbeitung der aus allen zwei Spalten (*sekunden*, *minuten*) summierten Werte in Sekunden. In Abhängigkeit der Bedingungen in Zeile 16 und 19 wird die Ausgabe entsprechend verändert. So entspricht die Ausgabe bei einen Wert unter 60 Sekunden entsprechend der Zeile 17. Dem Objekt *total_temp_s* wird der angezeigte String übergeben und dem Listenobjekt *totaltime* angehangen (Zeile 17, 18). Damit erscheint die Ausgabe unter dem ersten Listenelement *Gesprächszeit zurücksetzen* (Zeile 31) (Abbildung 5-32).

Abbildung 5-32: Ausgabe der Gesprächszeit

Entsprechend der Summe erfolgt die Anweisung für Umrechnung und Ausgabe für Minuten ab Zeile 19. Darin ist die Umrechnung in Minuten und für die restlichen Sekunden implementiert. Werden beide Bedingungen nicht wahr, so liegt die Summe oberhalb einer Stunde und die Anweisungen ab Zeile 25 mit den entsprechenden Umrechnungen werden ausgeführt. Am Ende werden beide Ergebnisse wieder dem Listenobjekt angefügt (Zeile 23, 28) und zur Ausgabe gebracht (Zeile 31). Der Grund für die Umkonvertierung der Werte in einen String mittels *str()* besteht in dem abgespeicherten Typ in der Datenbanktabelle. Dort werden die Werte als ganze Zahl *int* hinterlegt. Die Methode zur Ausgabe einer Auswahlliste *select()* erlaubt aber nur den Listenattribut-Typ String oder Unicode.

5.3.4 Personalisierung der Ausgabeboxen

Neben der Anzeige eigener Texte in verschiedenen Ausgabeboxen und Eingabefenstern kann die Position und die Textformatierung der Elemente personalisiert werden. Position, Struktur und Formatierung aller Elemente in XBMC sind in XML-Dateien abgelegt. Durch gezielte Parameteränderungen kann so der Entwickler das Aussehen von XBMC mitbestimmen. Im Systemkonzept soll die Benachrichtigungsbox, die Auswahlbox und die Dialogbox in der Position verändert beziehungsweise der Text formatiert werden. Lokalisiert sind die Dateien unter */usr/share/xbmc/addons/skin.confluence/720p*.

In der Datei *DialogKaiToast.xml* wird das Aussehen der Benachrichtigungsbox, die im Systemkonzept einen Zustand oder ein Ereignis signalisiert, festgelegt.

```
1 <coordinates>
```

```
2       <posy>50</posy>
```

Unter dem Element *coordinates* kann die Position der Box bestimmt werden. Für die Umpositionierung vom rechten unteren Rand an den rechten oberen Rand wird der *posy*-Parameter entsprechend verändert (Zeile 2).

```
3 <control type="fadelabel" id="401">
4       <description>Line 1 Label</description>
5 <textcolor>red</textcolor>
```

Einzelne Lines (Zeilen) einer Box sind in der XML-Struktur auf separate Elemente aufgeteilt. Damit kann der Entwickler jede Zeile einzeln editieren. Die Farbe des Textes in der erste Zeile der Box wird von weiß auf rot gesetzt (Zeile 5).

```
6 <control type="fadelabel" id="402">
7       <description>Line 2 Label</description>
8       <align>center</align>
```

Der Text in der zweiten Zeile wird von ehemalig linksbündig auf zentriert geändert (Zeile 8).

Aussehen und Position der Auswahlbox und der Dialogbox sind in DialogYesNo.xml beziehungsweise DialogOK.xml hinterlegt.

```
1 <control type="label" id="1">
2       <description>header label</description>
3       <align>center</align>
4 <control type="label" id="2">
5       <description>dialog line 1</description>
6       <align>center</align>
7 <control type="label" id="3">
8       <description>dialog line 2</description>
9       <align>center</align>
10 <control type=“label“ id=“4“>
11      <description>dialog line 3</description>
12      <align>center<align>
```

An den Boxen soll lediglich der Text in allen Zeilen zentriert zur Anzeige gebracht werden. Dafür sind in beiden Fällen unter den entsprechenden Label die *align*-Parameter zu ändern (Zeile 3, 6, 9, 12).

6 Bewertung und Fazit

In diesem Kapitel erfolgt eine Diskussion und Bewertung der Ergebnisse nach der Implementierung und eines umfangreichen Praxistests. Dabei sollen auf eventuelle Abweichungen zwischen den erwartenden und letztendlich erzielten Ergebnissen eingegangen werden. Das Fazit bildet den Abschluss des Kapitel's.

6.1 Bewertung der CTI-Anwendung

Mit TapiPhone stand dem Entwickler die bereits fertige Lösung einer CTI-Anwendung zur Verfügung. Für die Kommunikation mit dem Media Center XBMC musste hier noch die Implementierung eines Client- und Server-Socket sowie eine Ereignisbehandlung vorgenommen werden.

Bei der Implementierung des Server-Socket's zur Steuerung des Telefon's über XBMC mittels TAPI-Funktionen verlief die Umsetzung wie geplant. Der Socket wird beim Start des Programm's geöffnet und verbindet sich bei Assoziationen mit dem Client. Erst nach Beendigung des Programm's wird der Socket wieder geschlossen.

Im Verlauf der Implementierung des Client-Socket's traten für die Anrufsignalisierung und Zustandsbenachrichtigungen in XBMC einige Umsetzungsprobleme auf. Geplant war es, den Socket in jeden Call-Zustand innerhalb der Callback-Funktion neu mit dem Server-Socket zu verbinden und zu beenden. Dies hätte den Vorteil, dass sich bei einer frühzeitigen Beendigung der Verbindung der Socket ordnungsgemäß schließt. Mögliche Socket-Error's würden damit ausgeschlossen. Es stellte sich allerdings heraus, dass womöglich ein Timer-Problem innerhalb der Socket-Funktionen vorlag. Grund der Annahme bestand nach der Ausführung des Programm's im Debug-Modus. Während bei der normalen Abarbeitung des Programm's der Anschein erweckt wurde, der Socket befindet sich beim Aufruf des Senden's noch nicht im sendebereiten Zustand, verlief bei der Einzelschrittprozedur das Senden ohne Probleme. Dieser Sachverhalt beruht nur auf Spekulationen und bedarf sicherlich noch einer gründlichen Untersuchung. Aufgrund von Zeitproblemen war dies im Rahmen der Diplomarbeit nicht möglich. Zur Lösung des Problem's wurde die Überlegung getroffen, den Socket im Call-Zustand OFFERING zu öffnen und erst bei DISCONNECT beziehungsweise IDLE zu schließen. Diese Überlegung stellte sich nach der Implementierung auch als die Richtige heraus.

Darüber hinaus zeigte sich im Verlauf von vielen Tests, dass die Übermittlung der Rufnummer durch den TAPI-Treiber trotz gesendeter Rufnummer nicht immer ordnungsgemäß funktionierte. Diese Fehlfunktion tritt jedoch in sehr großen und unregelmäßigen Abständen auf, weshalb eine zuvor geplante Fehlersuche nicht mehr in Betracht kam. Es ist allerdings fast auszuschließen, dass das Problem auf Seiten der CTI-

Anwendung liegt. Demnach liegt das Problem mit hoher Wahrscheinlichkeit am TAPI-Treiber.

6.2 Bewertung der Add-on's in XBMC

Da das Systemkonzept einen möglichst verbrauchernahen Anwendungsfall vorsieht, sollte nach dem Bootvorgang das Media Center sofort gestartet werden. Doch hier zeigte sich entgegen den Erwartungen das erste Problem. Gleich zu Beginn und ohne eines eingehenden oder ausgehenden Anrufs signalisiert das System einen Skriptfehler. Dieser Fehler bezieht sich also auf das im Hintergrund laufende Skript. Die Log-Datei von XBMC brachte den Fehler „error: (99, 'Cannot assign requested address')“ zur Anzeige. Eine mögliche Fehlerursache könnte die fehlende Netzwerkverbindung zur Zeit der Initialisierung des Skriptes sein. Dadurch kann sich der Server-Socket nicht an die lokale IP-Adresse binden. Nun sieht die Lösung vor, erst nach dem Start des VDR-Frontend das Media Center zu starten. So hat das System Zeit, eine Netzwerkverbindung aufzubauen. Damit wird der Fehler erfolgreich umgangen.

Nach einer den Erfordernissen angepassten Einarbeitung in die Skriptsprache Python erfolgte die Programmierung der Add-ons. Dafür standen dem Entwickler die XBMC-spezifischen Module zur Beeinflussung der grafischen Oberfläche in XBMC bereit. Durch umfangreiche Tests im Vorfeld der Implementierung erfolgte der Umgang mit den Funktionen und Methoden der Module ohne nennenswerte Probleme. Dank detaillierter Erläuterungen zu den Funktionen und Methoden wurde der Umgang damit auch erleichtert. In der Praxis erfolgte die Ausgabe auf dem Bildschirm wie erhofft.

Bei der Implementierung des Client- beziehungsweise Server-Socket gab es auch hier keine nennenswerten Probleme. Zwar musste eine Lösung bezüglich nichtblockierter Socket's gefunden werden, in dem Modul asyncore aber eine den Anforderungen gerechte Lösung gefunden.

Hingegen gab es Probleme bei der Umsetzung der Datenbankanbindung für das Telefonbuch. Erste Überlegungen sahen vor, das Datenbanksystem MySQL zu verwenden. Da die Pakete für MySQL nachinstalliert werden mussten und dies auf Basis von Python 2.6.5 geschah, ergab das Zusammenspiel zwischen Python 2.4 unter XBMC, aber unter Python 2.6.5 erstellte MySQL-Pakete, Syntaxfehler an einigen Codestellen. So meldete der Interpreter z.B. einen Fehler bei der Übergabe des Datenbankpfades der connect()-Funktion. Obwohl der Pfad als String deklariert wurde, ergab die Interpretation dieser Codezeile einen Syntax-Error durch inkorrekte Zeichenkodierung. Keine Abhilfe brachte die Paketerstellung unter einer älteren Version von Ubuntu und Python 2.4. Aufgrund dieser Probleme viel die Entscheidung auf das Datenbanksystem SQLite3. Zwar erwiesen sich die standardmäßig installierten Pakete von SQLite3 unter Python 2.6.5 auch als inkompatibel zu Python 2.4. Allerdings brachten die Pakete unter der älteren Ubuntu-Version Abhilfe und die Anbindung des Add-on's an die Datenbank erfolgte reibungslos. Aus Sicht des Anwender's ist das Datenbanksystem SQLite3 auch die

bessere Lösung. So hat der Anwender alle Softwarekomponenten in einem System. Hingegen ist die die Anbindung an eine MySQL-Datenbank mit einem Mehraufwand an Hardware in Form eines MySQL-Server's verbunden.

Im weiteren Verlauf der Implementierung sowie einigen Testphasen des System's zeigten sich weitere kleine Umsetzungsprobleme. Demnach sollte Multithreading für eine parallele Befehlsabarbeitung dienen. Grund dafür lag in der Anrufsignalisierung und der angezeigten Auswahlbox (siehe Abbildung 5-19, Abbildung 5-23, Abbildung 5-24). Angedacht war es, die Box nach einen frühzeitigen Abbruch der Verbindung durch den Anrufer wieder auszublenden. Das Problem lag nun darin, dass der Interpreter auf ein Eingabeereignis in Form der Buttons Ja/Nein wartete. Damit war das Programm blockiert. Darauf hin erfolgte die Implementierung der kompletten Ereignisbehandlung in einen separaten Thread. Die Simulation für das Drücken des Enter-Button's über die entsprechende Built-In-Funktion wurde weiter im Eltern-Thread abgehandelt. Zwar lies sich die vermeintliche Lösung interpretieren, jedoch blockierte das Programm im separaten Thread und somit die Abarbeitung der Befehlsfolge im Eltern-Thread. Trotz diverser Untersuchungen konnte das Problem nicht gelöst werden. Anzunehmen ist, dass der eigene Python-Interpreter von XBMC kein Multithreading unterstützt. Entgegen dieser Vermutung lässt sich aber der Quellcode ohne Fehler interpretieren und eine asynchrone Socket-Implementierung durch das asyncore-Modul bewerkstelligen. Aufgrund der begrenzten Zeit lies sich auf diesen Gebiet keine weiteren Nachforschungen anstellen und somit blieb das Programm in seiner ursprünglichen Umsetzung erhalten.

Die Umsetzung des Gesprächszeitzähler's verlief auch ohne bestimmte Vorkommnisse. Jedoch stellte sich heraus, dass mit dieser Implementierung nur eine Erfassung der Zeit für ankommende Anrufe möglich war. Wählt der Anwender aus dem Telefonbuch einen Kontakt und die Verbindung kommt zustande, so wird die Gesprächszeit nicht aufgezeichnet. Zwar ermittelt das im Hintergrund laufende Skript „popup.py“ die Zeit bei Beendigung der Verbindung, doch nicht die Zeit bei der Verbindung beider Teilnehmer und demzufolge keine Zeitdifferenz. Da auch hier die Zeit für eine genaue Fehlerdiagnose nicht zur Verfügung stand, wurde die Umsetzung so belassen.

6.3 Fazit

Hauptziel der Konzeption war es, dem Anwender des freien und quell-offenen Media Center XBMC neben Live-TV und Internet-TV-Diensten wie Online Video-Portale auch Kommunikations-Dienste anzubieten. Zum Erreichen der Ziele waren verschiedene Schritte vorzunehmen und aufkommende Probleme zu lösen.

Auf Basis der Begriffsdefinitionen zur kommerziellen IPTV-Lösung und der Internet-TV-Lösung erfolgte ein Überblick zu Übertragungstechnologien und Protokollen in Rechner-Netzen unter Einbeziehung des DVB-IPI-Standard. Wichtige technische Aspekte wurden benannt und einer genauen Betrachtung unterzogen. Folglich wurde auf den heutigen

Stand der Technik im Bereich IPTV und Internet-TV eingegangen. Außerdem wurden typische Dienste in beiden Bereichen vorgestellt.

Anschließend fand im weiteren Kapitel eine Präzisierung der Aufgabenstellung statt. Es wurde der Begriff eines HTPC geklärt und dessen Anforderungen, Aufbaukonzepte und Softwarekomponenten erläutert. Auf Grundlage dessen wurde das XBMC Media Center mit seinen Funktions- und Leistungsmerkmalen präsentiert.

Nach der Präzisierung wurde das Systemkonzept vorgestellt. Ausgehend von den Anforderungen wurde das System konzipiert. Dabei erfolgte eine Beschreibung der geplanten Anwendungsfälle. Es wurde auf die benötigten Hard- und Software-Komponenten für ein funktionierendes Komplettsystem eingegangen. Zur Realisierung mussten vorweg technische Grundlagen geklärt und analysiert werden.

Basierend auf der Konzeption erfolgte die Einrichtung und Implementierung auf Seiten des CTI-Server und des HTPC. Eine ausführliche Schilderung der einzelnen Schritte von der Installation des Betriebssystem's über die Einrichtung und Konfiguration beider Instanzen wurden gegeben. Ausführliche Erläuterungen zu relevanten Quelltextabschnitten bei der Softwareeinrichtung von der CTI-Anwendung TapiPhone und den Add-ons in XBMC erfolgten ebenfalls.

Einen umfangreichen Praxistest wurde das System nach der Implementierung unterzogen. Abgesehen von den aufgezeigten Umsetzungsproblemen in 6.1 und 6.2 erfolgte der Test zufriedenstellend. Alle im Systemkonzept definierten Anforderungen konnten in der Praxis angewandt werden.

7 Ausblick

Selbstverständlich unterstehen Software-Anwendungen einer stetigen Weiterentwicklung. Möchte der Anwender auf dem neuesten Stand sein, muss das System aktualisiert oder neu installiert werden. Aufgrund dieser Tatsache sind programmiertechnische Anpassungen der Dienste an die neue Version oder veränderte Konfigurationseinstellungen unabdingbar.

Im Laufe der Implementierung des Systemkonzeptes brachte die Entwicklergemeinde rund um XBMC die neue Version 10.0 Dharma offiziell heraus. Das Hauptaugenmerk der Version liegt auf dem neuen Add-on-System, welches auch hier im Systemkonzept schon zum Einsatz kommt und ein Kriterium darstellte. Aufgrund dieser Tatsache ist das System auf eine bevorstehende Umstellung gut gerüstet. Veränderungen bringt Dharma in der Fernsteuerung über das Netzwerk. Die dafür zuständige HTTP-API (siehe 3.2.2.4) wird durch die JSON-RPC Schnittstelle ersetzt. JSON-RPC ist ein Standard für die Kommunikation über ein Client-Server-System. Ebenso wird nun das Datenbanksystem MySQL für speichern von Audio- und Videoinformationen innerhalb der Bibliothek genutzt (siehe 3.2.2.2). Die Liste der hinzugefügten oder geänderten Funktionalitäten ist lang. Es bleibt jedoch festzuhalten, dass die Veränderungen keinen unmittelbaren Einfluss auf die Funktionalität des bestehenden System's haben werden.

Natürlich lassen sich auch unabhängig von aktualisierten Versionen die implementierten Dienste weiter verbessern. So ist es z.B. sinnvoll, den Gesprächszeitzähler um die einzelnen Gesprächszeiten mit dazugehörigen Kontakt zu erweitern. Auf diese Weise hat der Anwender einen praktischen Überblick aller getätigten Telefonate und dazugehörigen Zeiten.

Mit der Arbeit wurden die Anforderungen an das System in Abhängigkeit des damaligen Entwicklungsstandes von XBMC umgesetzt. Durch Aktualisierungen in der Software oder durch Funktionsverbesserungen wird das derzeitige System in der kommenden Zeit ständiger Veränderungen unterzogen sein.

Glossar

Daemon

Daemon stellt in Unix oder unixartigen-Systemen spezifische Dienste im Hintergrund bereit. Interaktionen mit den Diensten erfolgen nur auf indirektem Weg. Typischerweise erfolgt die Kommunikation über Socket's oder Pipe's.

DHCP

Das Dynamic Host Configuration Protocol (DHCP) ist ein standardisiertes Netzwerkprotokoll. Es dient der automatischen Adresszuweisung an Clients durch einen Server. Der Client kann ohne einer manuellen Zuweisung die IP-Adresse, Subnetzmaske, Gateway und DNS-Server vom Server beziehen.

DLL

In DLL-Dateien wird jeglicher Programmcode, der von mehr als einer Anwendung benötigt werden könnte, in eine einzelne Datei geschrieben und in den Hauptspeicher geladen. So erfolgt eine Reduzierung des Speicherplatzes einer Anwendung im Hauptspeicher. Ebenso ist bei einer Aktualisierung des Programmcodes nur eine Datei zu bearbeiten. Alle Programme können in diesem Fall auf die aktualisierte Datei zugreifen.

DLNA

Die Digital Living Network Alliance (DLNA) ist eine internationale Vereinigung von Herstellern aus der Computer-, Unterhaltungselektronik- und Mobilfunk-Branche mit dem Ziel, Hersteller-übergreifende Kompatibilität sicherzustellen. Dafür werden Heimnetzwerkgeräte, Tragbare Geräte und Infrastrukturgeräte zertifiziert.

GPL

Die GNU General Public License (GPL) ist eine Freie-Software-Lizenz mit Copyleft für die Lizenzierung freier Software. Ihren Ursprung hat die Lizenz im GNU-Projekt, welches für die Entwicklung eines vollständig freien Betriebssystem gegründet wurde.

H.264/MPEG-4 AVC

Das H.264/MPEG-4 Advanced Video Coding (AVC) ist ein Standard für hocheffiziente Videokompression und wird hauptsächlich bei der Übertragung von HDTV angewendet. Die ITU-Bezeichnung lautet H.264 während die ISO den Standard unter MPEG-4 AVC führt.

HTTP-Protokoll

Das Hyper Text Transfer Protocol (HTTP) ist ein Anwendungsprotokoll und wird zur Kommunikation zwischen Client und Server genutzt. Ein Client sendet Anfragen in Form von HTTP-Requests an den Server und dieser antwortet mit HTTP-Responses.

IGMP-Protokoll

Das Internet Group Management Protocol (IGMP) gehört der IP-Netzwerkfamilie an und dient der Organsiation von Multicastgruppen. Das Protokoll ermöglicht IPv4-Multicasting. Unter IP-Multicasting wird der Versand von IP-Paketen an mehrere Stationen gleichzeit, unter Verwendung einer IP-Adresse, verstanden.

Metadaten

Metadaten enthalten Informationen über andere Daten. Bei den beschreibenen Daten handelt es sich meist um größere Datensammlungen wie datenbanken oder Dateien.

MHP

Multimedia Home Platform (MHP) wurde vom internationalen DVB-Projekt als Standard verabschiedet und spezifiziert die Übertragung und Darstellung interaktiver Inhalte im digitalen Fernsehen auf Basis der Programmiersprache Java.

Middleware

Unter Middleware wird die Verbindungsstelle zwischen eigenständigen Software-Anwendungen verstanden. Als Verbindungsstelle kommt ebenfalls Software zum Einsatz. Diese stellt Mechanismen für den Datenaustausch zwischen den Softwareanwendungen bereit. Netzwerkkommunikation, Koordination, eine hohe Ausfallsicherheit sowie Heterogenität sind für einen reibungslosen Datenaustausch zwischen verschiedenen Anwendungen erforderlich.

MMS

Das Microsoft Media Server Protocol (MMS) ist ein Protokoll der Anwendungsschicht und dient der Übertragung von Multimedia-Streams.

Multicast

Multicast ist eine Adressierungsart und bezeichnet eine Punkt-zu-einer Gruppe-Verbindung.

QoS

Quality of Service (QoS) ist eine Menge von Qualitätsanforderungen zwischen individuellen Anwendungen und Protokollen. Es beschreibt die Güte eines Kommunikationsdienstes aus Sicht der Anwender.

Repository

Zur Speicherung und Beschreibung von digitalen Objekten dient ein Repository. Dabei handelt es sich um ein verwaltetes Verzeichnis. Unterschiedliche Funktionen werden durch ein Repository abgedeckt. Häufig wird es zur Versionsverwaltung verwendet. Darin werden Quellcodedateien oder andere Dateien im Repository ausgecheckt (auf den Rechner eines Programmierers geladen) und anschließend nach der Bearbeitung wieder eingecheckt unter Protokollierung der Veränderungen. Typische Softwarevertreter sind SVN oder CVS. Eine weitere Funktion liegt im Software-Repository. Darin sind wiederum Programmpakete und zugehörige Metadaten wie Beschreibung, Abhängigkeits-informationen und Changelogs enthalten. Paketmanager erlauben das Installieren und Aktualisieren aus dem Repository heraus.

RSS

„RSS ist ein plattform-unabhängiges auf XML basierendes Format; entwickelt um Nachrichten und andere Web-Inhalte auszutauschen.“ (http://www.rss-verzeichnis.de/was-ist-rss.php) Wahlweise steht RSS für Really Simple Syndication, Rich Site Summary oder RDF Site Summary. Eingeführt durch Netscape in der Version 0.91 kam der Durchbruch durch ständige Weiterentwicklungen. Bis heute ist RSS in der Version 0.90, 0.91, 0.93, 0.94, 1.0, 2.0 erhältlich. Allerdings wurden die Versionen teilweise von unterschiedlichen Firmen und Entwicklern herausgegeben und sind somit untereinander inkompatibel. Wird ein RSS-Feed abonniert, so versorgt der RSS-Channel den Adressaten mit kurzen Informationsblöcken. Bereitgestellte Daten im RSS-Format werden RSS-Feeds genannt. Zum Einlesen diverser RSS-Feeds dienen Webbrowser oder spezielle Programme namens RSS-Reader oder Feedreader.

RTP-Protokoll

Das Real-Time Transport Protocol (RTP) wird zur Übertragung kontinuierlicher audiovisueller Daten (Streams) über IP-basierte Netze verwendet. Es dient der Paketierung, Kodierung und Versendung von Daten. Das Protokoll kann sowohl für Unicast- und Multicast-Adressierung eingesetzt werden. Üblicherweise wird es über UDP betrieben.

RTSP-Protokoll

Das Real-Time Streaming Protocol (RTSP) wird zur Steuerung einer kontinuierlichen audiovisuellen Übertragung von Daten (Streams) oder Software über IP-basierte Netze verwendet. RTSP ist ein textbasiertes Protokoll (ähnelt in vielen Dingen HTTP) und kann über UDP oder TCP übertragen werden. Allerdings werden keine Nutzdaten durch das Protokoll übertragen. Die Aufgabe besteht lediglich in der Steuerung der Datenströme.

Scraper

Scraper, besser bekannt als Web-Scraper, ist ein Software-Technik zum Auslesen fremder Inhalte aus Webseiten oder Datenbanken.

TCP-Protokoll

Das Transmission Control Protocol (TCP) ist ein verbindungsorientiertes, paketvermitteltes Schicht 4-Protokoll und verbindet zwei Hosts für die Datenübertragung miteinander. Dabei werden die Daten segmentiert und mit einen Header versehen, der neben der Quell- und Zieladresse auch eine Sequenznummer beinhaltet. Anhand dieser Nummer können die Segmente auf der Empfängerseite rekonstruiert werden. Der Empfänger quittiert jedes empfangene Paket. Mit dieser Methode werden fehlende Pakete erkannt und nochmalig angefordert.

UDP-Protokoll

Das User Datagram Protocol (UDP) ist ein verbindungsloses, paketvermitteltes Schicht 4-Protokoll der Internetfamilie. Aufgabe des Protokoll's ist es, die Daten der richtigen Anwendung zukommen zu lassen. Lediglich in der Adressierung durch Portnummern liegt die Aufgabe des Protokoll's. Die Datensicherung liegt außerhalb des Aufgabenbereiches um Verzögerungen bei der Datenübertragung zu minimieren. Da zwischen zwei Instanzen nicht erst eine Verbindung aufgebaut werden muss können die Partner schnell mit der Übertragung beginnen, ohne weiteren Overhead zu erzeugen. Bei verbindungslosen Übertragungen führen verloren gegangene Pakete zu keinen Sendeaussetzern, sondern lediglich zu einen Qualitätsverlust.

Unicast

Unicast ist eine Adressierungsart und bezeichnet eine Punkt-zu-Punkt-Verbindung.

UPnP

Der UPnP-Standard bedient sich einer Menge an Netzwerkprotokollen und Datenformaten für die herstellerübergreifende Ansteuerung in einem IP-Netzwerk befindlichen Geräten. Zur Verbindung können alle IP-unterstützen physikalischen Medien verwendet werden. UPnP-Clients finden durch bestimmte Mechanismen in der Adressierung, Lokalisierung, Beschreibung und Steuerung automatisch UPnP-fähige Geräte im Netzwerk. Gestört kann der Datenverkehr nur durch Firewalls oder nicht freigegebene Ports am Router werden.

VDPAU

Dank der Programmierschnittstelle Video Decode and Presentation API (VDPAU) kann das Dekodieren des Videostrom's und Nachbearbeiten des dekodierten Material's mit Hilfe der Hardwarebeschleunigung des Grafikprozessors erfolgen. Mit VDPAU übernimmt die Grafikkarte Berechnungen von Kompensation, Transformation und Coding und ermöglicht so eine flüssige Wiedergabe des Videomaterials.

Web 2.0

Allgemein wurde der Begriff Web 2.0 erstmalig Ende 2003 in einem Fachmagazin für IT-Manager vorgestellt und gegenüber einer breiten Öffentlichkeit erwähnt. Web 2.0 bezieht sich neben spezifischen Technologien oder Innovationen besonders auf die veränderte Nutzung des Internets. Eine Vielzahl von Nutzern erstellen, bearbeiten oder verteilen nun selbst Inhalte nach belieben über das Internet mit Hilfe interaktiver Anwendungen.

X11

X11 steht für X Window System und beinhaltet eine Sammlung von Protokollen, Computerprogrammen und Standards zur Ansteuerung grafischer Bildschirme im Allgemeinen. Vor allem unter Unix-Systemen dient es der Anzeige grafischer Benutzeroberflächen.

Literaturverzeichnis

[39] Weigand, Michael: Objektorientierte Programmierung mit Python, MITP-Verlag, 3., aktualisierte Auflage 2006

[13, 14] ETSI TR 102 033 (V1.1.1) „Digital Video Broadcasting (DVB); Architectural framework for the delivery of DVB-services over IP-based networks“

[16] ETSI TS 102 034 (V1.4.1) „Digital Video Broadcasting (DVB); Transport of MPEG-2 TS Based DVB Services over IP Based Networks"

[10] ITU-T H.720 „Overview of IPTV terminal devices and end systems“

[31] http://3dfusion.de/artikel/preview/Home_Theater_Personal_Computer/1_Einleitung_Die_pas sende_Hardware/

[25] http://breitbandinitiative.de/news/flaechendeckendes-breitband-in-2010

[11] http://de.wikipedia.org/wiki/Digital_Video_Broadcasting

[5] http://de.wikipedia.org/wiki/DVB-IPI

[30, 32, 34] http://de.wikipedia.org/wiki/Home_Theater_Personal_Computer

[4, 17, 18] http://de.wikipedia.org/wiki/Internet_Protocol_Television

[2, 3] http://de.wikipedia.org/wiki/Internetfernsehen

[1] http://de.wikipedia.org/wiki/IPTV

[33] http://de.wikipedia.org/wiki/Multifunktionsger%C3%A4t

[40] http://de.wikipedia.org/wiki/Python_%28Programmiersprache%29

[21] http://de.wikipedia.org/wiki/Streaming-Format

[20, 24] http://de.wikipedia.org/wiki/Streaming_Media

[23] http://de.wikipedia.org/wiki/MMS-Protokoll

[22] http://de.wikipedia.org/wiki/Streaming-Protokoll

[48] http://de.wikipedia.org/wiki/Telephony_Application_Programming_Interface

[47] http://de.wikipedia.org/wiki/Unified_Modeling_Language

[64] http://de.wikipedia.org/wiki/Video_Disk_Recorder

[37] http://de.wikipedia.org/wiki/XBMC_Media_Center

[56] http://docs.python.org/library/asyncore.html

[55] http://msdn.microsoft.com/en-us/library/3d46645f%28v=vs.80%29.aspx

[73] http://openbook.galileocomputing.de/python/python_kapitel_15_002.htm

[57] http://openbook.galileocomputing.de/python/python_kapitel_19_003.htm#mjea5883d439a425e2f974548b406b56c00

[53] http://openbook.galileocomputing.de/python/python_kapitel_20_001.htm

[49, 50, 51] http://telecom.htwm.de/telecom/praktikum/CTI_TAPI/CTI_TAPI.htm

http://telecom.htwm.de/telecom/praktikum/CTI_TAPI/images/TAPI-Konzept_screen.gif

[59, 70] http://vdr-free.de/wiki/index.php?title=FreeVDR_3.0

[71] http://wiki.ubuntuusers.de/Soundsystem

[45, 46] http://wiki.xbmc.org/index.php?title=Add-ons

[62] http://wiki.xbmc.org/index.php?title=Add-ons_for_XBMC_%28Developement%29

http://wiki.xbmc.org/index.php?title=File:Webserver.jpg

[35] http://wiki.xbmc.org/index.php?title=Introduction

[43] http://wiki.xbmc.org/index.php?title=Plugins

[38, 41, 42] http://wiki.xbmc.org/index.php?title=Python_Development

[44] http://wiki.xbmc.org/index.php?title=Scripts

[54] http://www.codeplanet.eu/tutorials/csharp/4-tcp-ip-socket-programmierung-in-csharp.html?start=4

[26] http://www.crn.de/datacenter/artikel-5532-3.html

[19, 27] http://www.crn.de/datacenter/artikel-5532-4.html

[8] http://www.etsi.org/WebSite/homepage.aspx

[9] http://www.ietf.org/

[29] http://www.iptv-anbieter.info/

[7] http://www.itu.int/en/ITU-T/gsi/iptv/Pages/default.aspx

[12] http://www.itwissen.info/definition/lexikon/digital-video-broadcasting-DVB.html

[15] http://www.itwissen.info/definition/lexikon/Settop-Box-STB-set-top-box.html

[28] http://www.netzwelt.de/news/82210-internet-fernseher-bieten-tv-hersteller.html

[69] http://www.vdr-wiki.de/wiki/index.php/Dvb-apps

[58, 68] http://www.vdr-wiki.de/wiki/index.php/FreeVDR

[60] http://www.vdr-wiki.de/wiki/index.php/LIRC

[67] http://www.vdr-wiki.de/wiki/index.php/Streaming

[66] http://www.vdr-wiki.de/wiki/index.php/Xine-plugin

[65] http://www.vdr-wiki.de/wiki/index.php/Xineliboutput-plugin

[36] http://xbmc.org/about/

[63] http://www.xfce.org/

[61, 72] https://launchpad.net/~henningpingel/+archive/xbmc

[52] https://www.telecom.hs-mittweida.de/index.php?eID=tx_nawsecuredl&u=0&file=fileadmin/verzeichnisfreigaben/telecom/winkler/vorlesungen/sockets.pdf&t=1295889766&hash=3153bf6f6e19c953b006e568c91818cc

[6] https://www.telecom.hs-mittweida.de/index.php?eID=tx_nawsecuredl&u=0&file=fileadmin/verzeichnisfreigaben/telecom/winkler/vorlesungen/standardisierung.pdf&t=1296814808&hash=eda71254b4cde2c8124c5477e3e28437

Anlagen

channels.conf

```
ZDF;ZDFmobil:482000:I999B8C23D12M16T8G4Y0:T:27500:545=2:546=deu@3,547=2ch@:551:0:514:8468:0:0
3sat;ZDFmobil:482000:I999B8C23D12M16T8G4Y0:T:27500:561=2:562=deu@3,563=2ch@3:567:0:515:8468:0:0
neo/KiKa;ZDFmobil:482000:I999B8C23D12M16T8G4Y0:T:27500:593=2:594=deu@3:599:0:517:8468:0:0
ZDFinfokanal;ZDFmobil:482000:I999B8C23D12M16T8G4Y0:T:27500:577=2:578=deu@3:551:0:516:8468:0:0
DasErste;ARD:506000000:B8C23D12G4M64T8Y0:T:0:1537=2:1538=deu@4,1539=deu@4:1543:0:96:8468:14849:0
Einsfestival;ARD:506000000:B8C23D12G4M64T8Y0:T:0:81=2:82=deu@4:87:0:5:8468:14849:0
arte;ARD:506000000:B8C23D12G4M64T8Y0:T:0:33=2:34=deu@4,35=fra@4:39:0:2:8468:14849:0
Phoenix;ARD:506000000:B8C23D12G4M64T8Y0:T:0:49=2:50=deu@4:55:0:3:8468:14849:0
MDR Sachsen;ARD:562000000:B8C23D12G4M64T8Y0:T:0:1553=2:1554=deu@3,1555=deu@3:1559:0:97:8468:15105:0
rbb Brandenburg;ARD:562000000:B8C23D12G4M64T8Y0:T:0:2833=2:2834=deu@3:2839:0:177:8468:15105:0
WDR Köln;ARD:562000000:B8C23D12G4M64T8Y0:T:0:4193=2:4194=deu@3:4199:0:262:8468:15105:0
Bayerisches FS Nord;ARD:562000000:B8C23D12G4M64T8Y0:T:0:529=2:530=deu@3:535:0:33:8468:15105:0
3sat;ZDFmobil:482000000:B8C23D12G4M16T8Y0:T:0:0:0:0:0:515:8468:514:0
neo/KiKa;ZDFmobil:482000000:B8C23D12G4M16T8Y0:T:0:0:0:0:0:517:8468:514:0
ZDF;ZDFmobil:482000000:B8C23D12G4M16T8Y0:T:0:0:0:0:0:514:8468:514:0
ZDFinfokanal;ZDFmobil:482000000:B8C23D12G4M16T8Y0:T:0:0:0:0:0:516:8468:514:0
```

script.popup → default.py

```
1 #Autor: Matthias Baier
2 #Skript zur Anrufsignalisierung und Anrufsteuerung in XBMC
3 #Asynchroner Server mit Datenbankabfrage
4
5 import socket,asyncore,sqlite3,re,time
6 from time import gmtime,strftime
7 import xbmc,xbmcgui
8
9 #Klassendefinition ServerSocket von Objektoberklasse "asyncore.dispatcher"
10 class ServerSocket(asyncore.dispatcher):
11     #Generierung eines neuen Objekts nach Bauplan der Klasse ServerSocket
12     #Konstruktormethode
13     def __init__(self,host,port):
14             asyncore.dispatcher.__init__(self)
15             #Socket einrichten
16             self.create_socket(socket.AF_INET,socket.SOCK_STREAM)
17             #Socket an Adresse und Port binden
18             self.bind((host,port))
19             #Socket wartet auf Verbindung (max. 1 Client)
20             self.listen(1)
21     #Methodendefinition zur Annahme der Clientverbindung
22     def handle_accept(self):
23             (clientsocket,address)=self.accept()
24             print "Verbunden mit", address
25             #Dispatcher starten
26             SocketHandler(clientsocket)
27
28 # Socket Handler starten
29 class SocketHandler(asyncore.dispatcher):
30     #Methodendefinition zum Empfang und Verarbeitung der Nachricht
31     def handle_read(self):
32             global zeit1
33             dialog=xbmcgui.Dialog()
34             self.buffer=self.recv(1024)
35             if len(self.buffer)>0:
36                     #Ruf angenommen
37                     if self.buffer=="connect":
38                             xbmc.executebuiltin("Notification(Eingehender Anruf,Anruf
angenommen,5000,/root/.xbmc/userdata/notifications/test3.jpg)")
39                             zeit1=time.clock()
40                             print "Zeit1",zeit1
```

```
41                      #Ruf beendet und optionale Wiedergabe
42                      elif self.buffer=="disconnect":
43                              xbmc.executebuiltin("Notification(Eingehender Anruf,Anruf
beendet,5000,/root/.xbmc/userdata/notifications/test2.jpg)")
44                              xbmc.executebuiltin("PlayerControl(Play)")

45                              if zeit1>0:
46                                      zeit2=time.clock()
47                                      print "Zeit2",zeit2
48                                      zeitdiff=zeit2-zeit1
49                                      if zeitdiff/60<1:
50                                              print "Zeitdifferenz_sekunden",zeitdiff
51                                              ok=dialog.ok('Gesprächsdauer','Die
Gesprächsdauer betrug %s Sekunden' % int(zeitdiff))
52                                      else:
53                                              zeitdiff_min=zeitdiff/60
54                                              zeitdiff_sec=zeitdiff%60
55                                              print "Minuten"

56                                              ok=dialog.ok('Gesprächsdauer','Die
Gesprächsdauer betrug %s Minuten und %s Sekunden' % (int(zeitdiff_min),int(zeitdiff_sec)))
57                                      ct=calltimeSQLite(zeitdiff)
58                                      zeit1=0
59                              else:
60                                      pass
61                              self.close()
62                      #Ruf nicht angenommen
63                      elif self.buffer=="onhook":
64                              xbmc.executebuiltin("XBMC.Notification(Eingehender
Anruf,Anruf wurde nicht angenommen,5000,/root/.xbmc/userdata/notifications/test2.jpg)")
65                              self.close()
66                      #Rufnummer unterdrückt
67                      elif self.buffer=="unknown":
68                              cur_time=getTime()
69                              xbmc.executebuiltin("XBMC.Notification(Eingehender
Anruf,Anruf von unbekannt,10000,/root/.xbmc/userdata/notifications/unknown_contact.jpg)")
70                              xbmc.executebuiltin("PlayerControl(Play)")
71                              ret=dialog.yesno("Anruf Optionen",cur_time+": Anruf von
unbekannt",","Wollen Sie den Anruf unterdrücken?")
72                              if ret==True:
73                                      getSocket('141.55.244.75',7777)
74                                      xbmc.executebuiltin("Notification(Eingehender
Anruf,Anruf unterdrückt,5000,/root/.xbmc/userdata/notifications/test2.jpg)")
```

```
75                                      xbmc.executebuiltin("PlayerControl(Play)")

76                              elif ret==False:
77                                      pass
78                      else:
79                              #Rufnummer gesendet
80                              nmb=self.buffer
81                              #Rufnummer auf x*0 prüfen
82                              if "0" in self.buffer[0:1]:
83                                      nmb=self.buffer[1:]
84                              else:
85                                      pass
86                              complcontact=getSQLite3Access(nmb)
87                              cur_time=getTime()
88                              #Ausgabe der Nachricht auf OSD und optionale
Filmunterbrechung
89                              xbmc.executebuiltin("XBMC.Notification(Eingehender
Anruf,Anruf von %s,10000,%s)" % (complcontact[1],complcontact[2]))
90                              xbmc.executebuiltin("PlayerControl(Play)")
91                              ret=dialog.yesno("Anruf Optionen",cur_time+": Anruf von
"+complcontact[1],","Wollen Sie den Anruf unterdrücken?")
92                              if ret==True:
93                                      getSocket('141.55.244.75',7777)
94                                      xbmc.executebuiltin("Notification(Eingehender
Anruf,Anruf unterdrückt,5000,/root/.xbmc/userdata/notifications/test2.jpg)")
95                                      xbmc.executebuiltin("PlayerControl(Play)")

96                              elif ret==False:
97                                      pass
98                              if complcontact[0]==1:
99                                      yesno=dialog.yesno("Telefonbuch","Kontakt in
Telefonbuch eintragen?")
100                                     if yesno==True:
101                                             c=createContact(nmb)
102                                     else:
103                                             pass
104                             elif complcontact[0]==0:
105                                     pass
106
107             else:
108                     self.close()
109     #pass um Warnung "unhandled write,close event" zu unterdrücken
110     def handle_write(self):
111             pass
```

```
112     def handle_close(self):
113             self.close()
114
115 #Funktion zum Einrichten des ClientSocket für Anrufablehnung
116 def getSocket(host,port):
117     csock=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
118     csock.connect((host,port))
119     csock.send("dropcall")
120     csock.close()
121
122 #Funktion zur Datenbankabfrage für ankommenden Anruf
123 def getSQLite3Access(nmb):
124     complcontact=[]
125     connection=sqlite3.connect("/root/.xbmc/userdata/Database/TelefonbuchHtwm.db")
126     cursor=connection.cursor()
127     print nmb
128     cursor.execute("SELECT vorname,nachname,kontaktbild FROM telefonbuch WHERE
rufnummer=?",[nmb])
129     rows=cursor.fetchall()
130     print rows
131     #Kontakt vorhanden?
132     if not rows:
133             #Übermittlung der Rufnummer
134             complcontact.append(1)
135             complcontact.append(nmb)
136
        complcontact.append("/root/.xbmc/userdata/notifications/unknown_contact.jpg")
137     else:
138             #Übermittlung des Namen
139             complcontact.append(0)
140             for row in rows:
141                     contact=row[0]+' '+row[1]
142                     conpic=row[2]
143             #kein Kontaktbild vorhanden?
144             if not row[2].endswith('.jpg') or row[2].endswith('.png') or
row[2].endswith('.gif'):
145                     conpic="/root/.xbmc/userdata/notifications/empty_contact.jpg"
146             else:
147                     pass
148             complcontact.append(contact)
149             complcontact.append(conpic)
150     cursor.close()
151     connection.close()
152     print complcontact
```

```
153     return complcontact
154 
155 #Funktion zur aktuellen Zeitermittlung
156 def getTime():
157     cur_time=strftime("%d %b %Y um %H:%M:%S",gmtime())
158     print cur_time
159     return cur_time
160 
161 #Funktion zum Einrichten eines neuen Kontakts in Datenbank
162 def createContact(nmb):
163     print nmb
164     dialog=xbmcgui.Dialog()
165     connection=sqlite3.connect("/root/.xbmc/userdata/Database/TelefonbuchHtwm.db")
166     cursor=connection.cursor()
167     print "Verbunden mit TelefonbuchHtwm"
168     #Kontaktdaten eingeben
169     kb=xbmc.Keyboard(nmb,'Rufnummer übernehmen?',False)
170     kb.doModal()
171     if (kb.isConfirmed()):
172             ruf=kb.getText()
173             print ruf
174     kb=xbmc.Keyboard('','Bitte geben Sie den Vornamen ein',False)
175     kb.doModal()
176     if (kb.isConfirmed()):
177             vor=kb.getText()
178             print vor
179     kb=xbmc.Keyboard('','Bitte geben Sie den Nachnamen ein',False)
180     kb.doModal()
181     if (kb.isConfirmed()):
182             nach=kb.getText()
183             print nach
184     picfile=dialog.browse(1,'Bitte Pfad zum Kontaktbild
auswählen','myprograms','.jpg|.png',True,False,'/root/.xbmc/userdata/notifications/Kontaktbilde
r')
185     print picfile
186     text=(vor+' '+nach+' '+ruf+' '+picfile)
187     liste=re.split(r"\s",text)
188     print liste
189     sql="INSERT INTO telefonbuch VALUES (?,?,?,?)"
190     cursor.execute(sql,liste)
191     yesno=dialog.yesno('Telefonbucheintrag','Kontakt wirklich zum Telefonbuch
hinzufügen?')
192     #Kontakt abspeichern?
193     if yesno==True:
```

```
194                 connection.commit()
195                 xbmc.executebuiltin('Notification(Telefonbuch,Kontakt wurde
hinzugefügt,5000,/root/.xbmc/userdata/notifications/db_add.png)')
196                 cursor.close()
197                 connection.close()
198     else:
199                 cursor.close()
200                 connection.close()
201
202 #Funktion zum Datenbankeintrag der Gesprächszeit
203 def calltimeSQLite(zeitdiff):
204     connection=sqlite3.connect("/root/.xbmc/userdata/Database/Gespraechsdauer.db")
205     cursor=connection.cursor()
206     if zeitdiff/60<1:
207                 time=(int(zeitdiff),0)
208                 cursor.execute("INSERT INTO gespraechszeit VALUES (?,?)",time)
209     else:
210                 zeitdiff_min=zeitdiff/60
211                 zeitdiff_sec=zeitdiff%60
212                 time=(int(zeitdiff_sec),int(zeitdiff_min))
213                 cursor.execute("INSERT INTO gespraechszeit VALUES (?,?)",time)
214     connection.commit()
215     cursor.close()
216     connection.close()
217
218 #Generierung des Sockets
219 s=ServerSocket('141.55.243.248',6666)
220 #Socket Polling
221 asyncore.loop(1)
```

script.program.calltime → default.py

```
1 #Autor: Matthias Baier
2 #Skript für das Gesprächszeit-Add-on
3
4 import sqlite3
5 import xbmc,xbmcgui
6
7 calltime_sec=[]
8 calltime_min=[]
9 totaltime=["Gesprächszeit zurücksetzen"]
10 dialog=xbmcgui.Dialog()
11 connection=sqlite3.connect("/root/.xbmc/userdata/Database/Gespraechsdauer.db")
12 cursor=connection.cursor()
```

```
13 print "Verbunden mit Gespraechsdauer"
14 cursor.execute("SELECT sekunden,minuten FROM gespraechszeit")
15 rows=cursor.fetchall()
16 for row in rows:
17     sec=row[0]
18     minu=row[1]
19     calltime_sec.append(sec)
20     calltime_min.append(minu)
21 print calltime_sec
22 print calltime_min
23 addtime_sec=sum(calltime_sec)
24 print addtime_sec
25 addtime_min=sum(calltime_min)
26 print addtime_min
27 addtime_msec=addtime_min*60
28 min_sec=addtime_msec+addtime_sec
29 print min_sec
30 #Umrechnung der Zeiteinheiten
31 if min_sec/60<1:
32     total_temp_s=str(min_sec)+' Sekunden '
33     totaltime.append(total_temp_s)
34 if min_sec/60>1 and min_sec/3600<1:
35     total_min=min_sec/60
36     total_sec=min_sec%60
37     total_temp_m=str(total_min)+' Minuten : '+str(total_sec)+' Sekunden '
38     totaltime.append(total_temp_m)
39 elif min_sec/60>1 and min_sec/3600>1:
40     total_h=min_sec/3600
41     total_min=total_h/60
42     total_sec=total_h%60
43     total_temp_h=str(total_h)+' Stunden :'+str(total_min)+' Minuten : '+str(total_sec)+'
Sekunden '
44     totaltime.append(total_temp_h)
45 print totaltime
46 #Zeit ausgeben
47 ret=dialog.select("Gesprächszeit",totaltime)
48 #Zeit zurücksetzen
49 if ret==0:
50     cursor.execute("DELETE FROM gespraechszeit")
51     yesno=dialog.yesno('Telefonbucheintrag','Gesprächszeit wirklich zurücksetzen?')
52     #Bestätigung
53     if yesno==True:
54             connection.commit()
```

```
55                  xbmc.executebuiltin('Notification(Gesprächszeit,Gesprächszeit wurde
zurückgesetzt,5000,/root/.xbmc/userdata/notifications/db_del.png)')
56                  cursor.close()
57                  connection.close()
58      else:
59                  cursor.close()
60                  connection.close()
```

script.program.telephonebook → default.py

```
1 #Autor: Matthias Baier
2 #Skript für das Telefonbuch
3
4 import socket,sqlite3,re
5 import xbmc, xbmcgui
6
7 #Klassendefinition ClientSocket
8 class ClientSocket(object):
9       def __init__(self,host,port,callid):
10                  print "ClientSocket initialisiert"
11                  print callid
12                  csock=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
13                  csock.connect((host,port))
14                  print "Verbunden mit",csock
15                  csock.send("makecall:"+callid)
16                  csock.close()
17
18 dialog=xbmcgui.Dialog()
19 connection=sqlite3.connect("/root/.xbmc/userdata/Database/TelefonbuchHtwm.db")
20 cursor=connection.cursor()
21 print "Verbunden mit TelefonbuchHtwm"
22 cursor.execute("SELECT vorname,nachname,rufnummer,kontaktbild FROM telefonbuch")
23 rows=cursor.fetchall()
24 contacts=['Kontakt hinzufügen','Kontakt löschen']
25 numbers=[]
26 conpics=[]
27 for row in rows:
28      contact=row[0]+' '+row[1]
29      number=row[2]
30      conpic=row[3]
31      contacts.append(contact)
32      numbers.append(number)
33      conpics.append(conpic)
34 print contacts
```

```
35 print numbers
36 print conpics
37 ret=dialog.select("Bitte Kontakt auswählen",contacts)
38 #Kontakt hinzufügen
39 if ret==0:
40     #Kontaktdaten eingeben
41     kb=xbmc.Keyboard('','Bitte geben Sie den Vornamen ein',False)
42     kb.doModal()
43     if (kb.isConfirmed()):
44             vor=kb.getText()
45             print vor
46     kb=xbmc.Keyboard('','Bitte geben Sie den Nachnamen ein',False)
47     kb.doModal()
48     if (kb.isConfirmed()):
49             nach=kb.getText()
50             print nach
51     kb=xbmc.Keyboard('','Bitte geben Sie die Rufnummer ein',False)
52     kb.doModal()
53     if (kb.isConfirmed()):
54             ruf=kb.getText()
55             print ruf
56     picfile=dialog.browse(1,'Bitte Pfad zum Kontaktbild
auswählen','myprograms','.jpg|.png',True,False,'/root/.xbmc/userdata/notifications/Kontaktbilde
r')
57     print picfile
58     text=(vor+' '+nach+' '+ruf+' '+picfile)
59     liste=re.split(r"\s",text)
60     print liste
61     sql="INSERT INTO telefonbuch VALUES (?,?,?,?)"
62     cursor.execute(sql,liste)
63     yesno=dialog.yesno('Telefonbucheintrag','Kontakt wirklich zum Telefonbuch
hinzufügen?')
64     #Bestätigung
65     if yesno==True:
66             connection.commit()
67             xbmc.executebuiltin('Notification(Telefonbuch,Kontakt wurde
hinzugefügt,5000,/root/.xbmc/userdata/notifications/db_add.png)')
68             cursor.close()
69             connection.close()
70     else:
71             cursor.close()
72             connection.close()
73 #Kontakt wählen
74 if ret>=2:
```

```
75      callid=numbers[ret-2]
76      callcon=contacts[ret]
77      callpic=conpics[ret-2]
78      print callcon
79      print callpic
80      if not callpic.endswith('.jpg') or callpic.endswith('.png') or callpic.endswith('.gif'):
81                              callpic="/root/.xbmc/userdata/notifications/empty_contact.jpg"
82      else:
83              pass
84      #Kontakt wird angerufen
85      c=ClientSocket('141.55.244.75',7777,callid)
86      xbmc.executebuiltin('Notification(Ausgehender Anruf,%s wird angerufen,10000,%s)' %
(callcon,callpic))
87 #Kontakt löschen
88 if ret==1:
89      cursor.execute("SELECT vorname,nachname FROM telefonbuch")
90      rows=cursor.fetchall()
91      contacts=[]
92      for row in rows:
93              contact=row[0]+' '+row[1]
94              contacts.append(contact)
95      print contacts
96      ret=dialog.select("Kontakt löschen",contacts)
97      removeid=numbers[ret]
98      cursor.execute("DELETE FROM telefonbuch WHERE rufnummer=?",[removeid])

99      yesno=dialog.yesno('Telefonbucheintrag','Kontakt aus dem Telefonbuch löschen?')
100     #Bestätigung
101     if yesno==True:
102             connection.commit()
103             xbmc.executebuiltin('Notification(Telefonbuch,Kontakt wurde
gelöscht,5000,/root/.xbmc/userdata/notifications/db_del.png)')
104             cursor.close()
105             connection.close()
106     else:
107             cursor.close()
108             connection.close()
```

Auszug aus TapiPhone → TapiPhoneDlg.cpp

```
1 // CTapiPhoneDlg-Meldungshandler
2
3 BOOL CTapiPhoneDlg::OnInitDialog()
4 {
```

```
5       ...
6       sendSocket = new CMyAsyncSocket(this);
7       serverSocket = new MyAsyncServerSocket(this);
8       //Server-Socket einrichten
9       serverSocket->Create(7777,1,FD_READ | FD_WRITE | FD_OOB | FD_ACCEPT |
FD_CONNECT | FD_CLOSE,(CString)"192.168.1.50");
10      iErrorCode = serverSocket->GetLastError();
11      serverSocket->Listen();
12      ...
13 }
14 //Callback-Funktion mit Ereignisbehandlung
15 void CTapiPhoneDlg::CallbackFunction(DWORD hDevice,DWORD dwMsg,DWORD
dwParam1,DWORD dwParam2,DWORD dwParam3)
16 {
17      LINECALLINFO *callInfo = new LINECALLINFO;
18      CString number;
19      CTime AktuelleZeit;
20      CString strZeit;
21      const char disc[] = "disconnect";
22      const char conn[] = "connect";
23      const char unkn[] = "unknown";
24      const char idle[] = "onhook";
25
26      //Ländercode für die Zeitanzeige auf Deutsch setzen
27      setlocale(LC_ALL, "German");
28
29      switch(dwMsg){
30      case LINE_CALLSTATE:
31              AktuelleZeit=CTime::GetCurrentTime();
32              strZeit=AktuelleZeit.Format("%d.%b %Y %H:%M:%S");
33              switch(dwParam1){
34              case LINECALLSTATE_ACCEPTED:
35                      //Ruf wurde angenommen
36                      m_CallInfoStrg.InsertString(-1,strZeit+(CString)": Freizeichen erkannt");
37                      break;
38              case LINECALLSTATE_RINGBACK:
39                      // Freizeichen wurde erkannt
40                      m_CallInfoStrg.InsertString(-1,strZeit+(CString)": Die gewählte Nummer
wird gerufen");
41                      break;
42              case LINECALLSTATE_BUSY:
43                      // Besetztzeichen wurde erkannt
44                      m_CallInfoStrg.InsertString(-1,strZeit+(CString)" Nummer ist besetzt");
45                      break;
```

```
46              case LINECALLSTATE_OFFERING:
47                      //Client-Socket verbinden
48                      sendSocket->Create();
49                      sendSocket->Connect((CString)"141.55.243.248", 6666);
50                      //eingehender Call
51                      m_hCall=(HCALL)hDevice;
52                      number=GetPhoneNumber(m_hCall);
53                      //Rufnummer mitgesendet?
54                      if(number!=""){
55                              m_CallInfoStrg.InsertString(-1,strZeit+((CString)": Anruf von ")
+m_pPhoneBook->m_vorname+((CString)" ")+m_pPhoneBook->m_nachname);
56                              char str[100];
57                              CT2CA _number(number);
58                              strcpy(str,_number);
59                              sendSocket->sendData(str);
60                      }
61                      else{
62                              m_CallInfoStrg.InsertString(-1,strZeit+((CString)": Anruf von ")
+number);
63                              sendSocket->sendData(unkn);
64                      }
65                      break
66              case LINECALLSTATE_CONNECTED:
67                      //Verbindung hergestellt
68                      m_CallInfoStrg.InsertString(-1,strZeit+(CString)" Verbindung wurde
hergestellt");
69                      sendSocket->sendData(conn);
70                      break;
71              case LINECALLSTATE_IDLE:
72                      //Ruf existiert aber keine Verbindung zustande gekommen
73                      m_CallInfoStrg.InsertString(-1,strZeit+(CString)" Verbindung
abgebrochen");
74                      sendSocket->sendData(idle);
75                      sendSocket->Close();
76                      break;
77              case LINECALLSTATE_DISCONNECTED:
78                      //Verbindung beendet
79                      m_CallInfoStrg.InsertString(-1,strZeit+(CString)" Verbindung wurde
beendet");
80                      sendSocket->sendData(disc);
81                      //Client-Socket schließen
82                      sendSocket->Close();
83                      break;
84              }
```

```
85      }
86 }
87
88 //CTI-Funktionalität für Rufnummer wählen
89 void CTapiPhoneDlg::OnBnClickedCall()
90 {
91      UpdateData(true);
92      //TAPI-Funktion
93      m_result = lineMakeCall(m_hLine,&m_hCall,m_PhoneNumberValue,0,NULL);
94
95      if(m_result<0){
96                m_CallInfoStrg.InsertString(-1,(CString)"Fehler beim Call");
97      }
98 }
99 //CTI-Funktionalität für Hörer auflegen
100 void CTapiPhoneDlg::OnBnClickedDrop()
101 {
102      //TAPI-Funktion
103      m_result = lineDrop(m_hCall,NULL,NULL);
104
105      if(m_result<0){
106                m_CallInfoStrg.InsertString(-1,(CString)"Error DropLine");
107      }
108 }
```

TapiPhone → MyAsyncSocket.cpp

```
1 // MyAsyncSocket.cpp: Implementierungsdatei
2
3 #include "stdafx.h"
4 #include "TapiPhone.h"
5 #include "MyAsyncSocket.h"
6 #include <string>
7
8 // CMyAsyncSocket
9 CMyAsyncSocket::CMyAsyncSocket(CTapiPhoneDlg *dlg)
10 {
11      canSend=true;
12      canReceive=false;
13      g_dlg=dlg;
14 }
15 CMyAsyncSocket::~CMyAsyncSocket()
16 {
17 }
```

```
18 // CMyAsyncSocket-Memberfunktionen
19 void CMyAsyncSocket::OnReceive(int nErrorCode) {
20     canReceive=true;
21     receiveData();
22 }
23 //CMyAsyncSocket-Memberfunktionen
24 void CMyAsyncSocket::OnSend(int nErrorCode){
25     canSend=true;
26 }
27 //CMyAsyncSocket-Memberfunktionen
28 void CMyAsyncSocket::sendData(const char *text){
29     if(canSend==true){
30             Send(text,strlen(text));
31     }
32 }
33 //CMyAsyncSocket-Memberfunktionen
34 void CMyAsyncSocket::receiveData(){
35     char buffer[1024];
36     int nRead;
37     CString message,number;
38     if(canReceive==true){
39             //Nachricht empfangen
40             nRead = Receive(&buffer,1024);
41             if(nRead==0) {
42                     Close();
43             }
44             else {
45                     for(int i=0;i<nRead;i++){
46                             message+=buffer[i];
47                     }
48                     //Nachricht für Hörer auflegen und Funktion aufrufen
49                     if (message.Find((CString)"dropcall")==0){
50                             g_dlg->OnBnClickedDrop();
51                     }
52                     //Nachricht für Kontakt wählen und Funktion aufrufen
53                     if (message.Find((CString)"makecall")==0){
54                             number=message.Mid(9);
55                             g_dlg->m_PhoneNumberValue=number;
56                             g_dlg->UpdateData(false);
57                             g_dlg->OnBnClickedCall();
58                     }
59                     else {
60                             Close();
61                     }
```

```
62                 }
63         }
64 }
```

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, den 21.02.2011

Matthias Baier